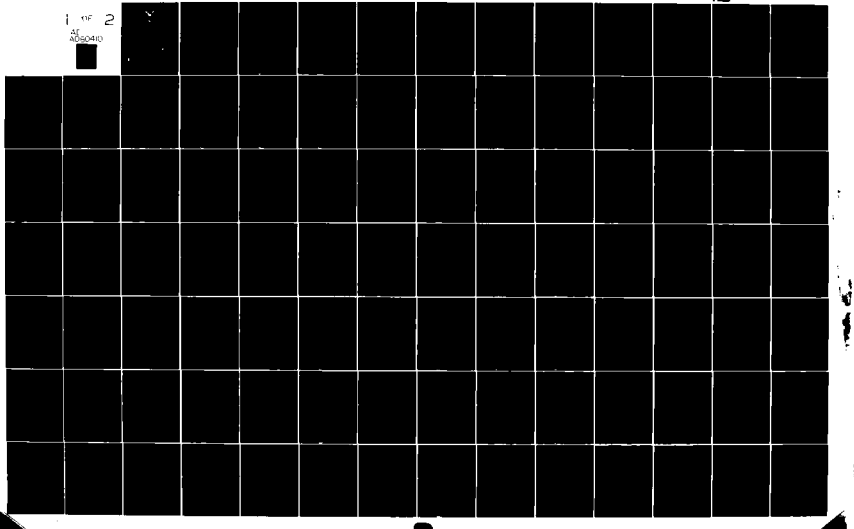
AD-A080 410 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A GRAPH MODEL REPRESENTATION OF A DISTRIBUTED PROCESSOR COMPUTE--ETC(U)
DEC 79 L A PALMER

UNCLASSIFIED AFIT/GCS/EE/79-12 NL

1 OF 2
AD
A080410



(14)

AFIT/GCS/EE/79-12

(6)

A GRAPH MODEL REPRESENTATION OF A DISTRIBUTED PROCESSOR COMPUTER SYSTEM.

THESIS

AFIT/GCS/EE/79-12

(10) Leslie A. Palmer
2Lt USAF

(9) Masters thesis,

(11) Dec 79

(12) 150

ϕ12225

AFIT/GCS/EE/79-12

A GRAPH MODEL REPRESENTATION OF A DISTRIBUTED PROCESSOR COMPUTER SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Leslie A. Palmer
2Lt USAF
Graduate Electrical Engineering
December 1979

Approved for public release; distribution unlimited

Preface

Although a lot of theory about graph models has been developed, relatively little work has been directed to the application of graph model techniques. This report describes the development of a graph model representation of the Digital Avionics Information System (DAIS), a distributed processor computer architecture. The class of graph models used in this investigation are known as Evaluation nets. Evaluation nets were developed by Dr. Gary Nutt and are one of the few graph model classes developed specifically to support computer system performance analysis efforts.

Thanks are due to Capt. Walt Seward who sponsored this investigation. Dr. Gary Lamont was most helpful as my advisor, providing guidance and helping solve some difficult problems during model construction. Special thanks are due to Delores Lamont who spent many long nights at her typewriter preparing the final copy of this report. Most of all, I am forever grateful to my wife, Linda. Without her personal sacrifices and continual encouragement, I would not have completed this project.

Contents

List of Figures

Abstract

Three evaluation net graph models of the Digital Avionics Information System (DAIS) were constructed. The three models represent three increasingly lower levels of detail; the third model represents the DAIS at a task flow level of detail. The models are evaluated as analysis tools. Methods are presented for analyzing the DAIS structure and performance and examples are given. The biggest problem associated with a performance analysis using evaluation nets is the recording of data collected during model analysis. A solution to the data recording problem is to utilize data automation techniques.

List of Abbreviations

BCIU	Bus Control Interface Unit
BCM	Bus Control Module
DAIS	Digital Avionics Information System
DMA	Direct Memory Access
E-net	Evaluation net
MTU	Multiplex Terminal Unit
PIM	Processor Interface Module
PIO	Programmed Input/Output
r-location	Resolution location
RT	Remote Terminal

A GRAPH MODEL REPRESENTATION OF A DISTRIBUTED PROCESSOR COMPUTER SYSTEM

I Introduction

Every computer system is designed to perform a certain function or set of functions. Whether or not a system performs its function(s) and, if so, how well does the system perform its function(s) are questions which lead to a performance evaluation effort. The most accurate way to evaluate a system's performance is to measure the system under its real workload (Ref 32:19). Very often, though, it is not feasible or practical to measure an existing system (Ref 5:12). When the system to be analyzed is not available for direct analysis, a model of the system can serve as the subject of analysis. Even if the system is available, there are many possible advantages of analyzing models instead of the real systems, such as cost-effectiveness, flexibility, and forecasting (Ref 12:10-11). This investigation focuses on the use of a particular class of graph models, known as Evaluation-Nets, to model a distributed processor computer system in support of a performance analysis effort.

DAIS

The Digital Avionics Information System (DAIS) is a distributed processor computer system developed for the Air Force to simulate avionics computer systems. Located at the Avionics Lab, Wright-Patterson Air Force Base, Ohio, the primary mission of the DAIS is to provide a host computer system on which to test and evaluate new or changes to existing avionics computer systems (Ref 33:1). The DAIS provides sufficient modularity and redundancy to allow a user to simulate all or a subset of the avionics computer system's functions (navigation, target acquisition, flight plan, on-board stores, etc.). Sensors, analog devices, weapons, and displays can be connected to the DAIS via a standardized multiplexed data bus system, MIL-STO-1553A (Ref 10,30,31).

At the time of this investigation, the DAIS has not yet been fully implemented (PDP-11/40 computers are being used to simulate AN/AYK-15 avionics processors and a DEC-10 computer system is being used to simulate the rest of the DAIS system and DAIS environment). For a general description of the DAIS, the reader is directed to Chapter II of this report. For a detailed description of the DAIS, the reader is referred to References 1,2,3,9,18,23, and 25.

Graph Models

A functional model describes how a system operates and can be used to derive a performance model in support of a performance analysis. Liba Svobodova (Ref 32) defines four classes

of functional models: flowchart models, parallel nets, finite-state models and queueing models (32:31). The first three classes are types of directed graph models in which, generally, nodes represent tasks and arcs represent flow of control.

Flowchart models, as the name implies, are derived from functional flowcharts and pictorally represent the computational tasks of a system. Given the execution time of the tasks and the probability of following different arcs, a simulation of system execution can be effected and analyzed. However, since Flowchart models do not provide the capability to represent amounts and types of resources or the scheduling of these resources, Flowchart models cannot be used to represent a total system, but only the programs executed by the processor (Ref 32:31).

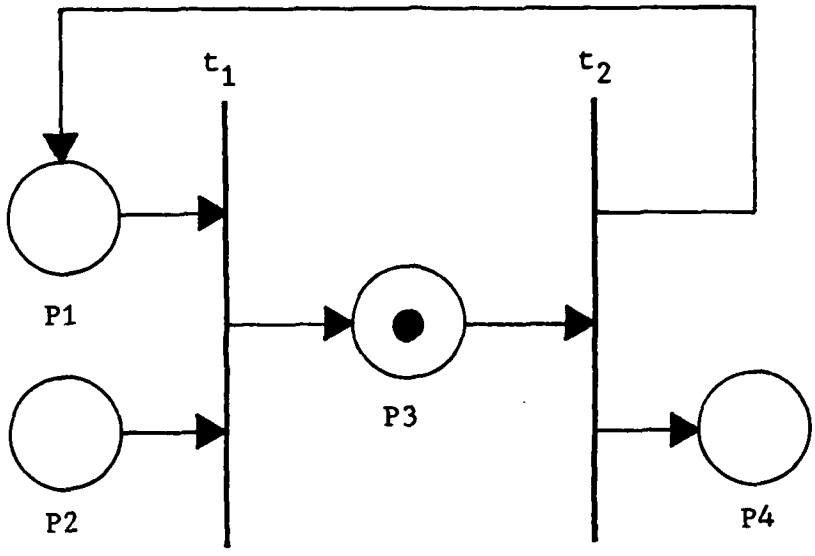
Nodes represent states of the system and arcs represent transitions between states in a finite-state model (Ref 32:32). Since a system state is defined by the states of the system components, parallel processing can be represented in a finite-state model. At the task level, though, the number of possible states of the system can be a large number. For example, in the DAIS system, a task can be in any one of six states and there are three general classes of tasks. For a two processor, two channel DAIS simulation, a finite-state model of $(6 \times 3 \times 2^4) = 288$ nodes would be required. (Addition of just one more processor and channel requires a model with over 4600 nodes!) The point here is that finite-state models are useful for high level analysis of a system, but may be quite

complex and unwieldy at a detailed task level.

Parallel nets are directed graphs with two types of nodes, transitions and places (Ref 32:32). Places represent the conditions which must be satisfied for a transition to be enabled. Transitions represent events which, when enabled, may occur. Tokens are markers on places which indicate whether or not the corresponding conditions are satisfied (a place with a token represents a satisfied condition). Places are only connected to transitions, and transitions are only connected to places, via arcs. Since several transitions may be enabled simultaneously and independently, parallel nets may describe concurrent, asynchronous processing (as in the DAIS). A simple parallel net model is shown in Figure 1. A token, represented by the solid black dot, resides on place p_3. This is interpreted as "CPU busy", and transition t_2 is enabled. When transition t_2 fires, t removes a token from p_3 and places a token on each of the places p_4 and p_1 (transition t_1 is now enabled).

Petri nets, Timed Petri nets, Colored Petri nets, and Evaluation nets are four classes of directed graph models that fall into Svobodova's parallel net category of functional models. Petri nets were used to derive the latter three classes of graph models just mentioned. Figure 1 could be interpreted as a graph representation of a Petri net.

Properties of Petri nets which make them useful for modeling distributed processor systems are their concurrency or



● : token

PLACES

P1 : CPU idle

P2 : Request for CPU

P3 : CPU busy

P4 : task complete

TRANSITIONS

t_1 : Allocate CPU

t_2 : Deallocate CPU

Fig. 1 Parallel Net Model (Ref 32:33)

parallelism and asynchronous nature (Ref 21:229). However, some properties of Petri nets limit, if not prevent, the use of Petri nets for performance analysis. Once a transition is enabled in a Petri net, the only action guaranteed is that the transition will fire within an unspecified but finite, future time interval. In addition, the firing of the transition takes zero time. Therefor it is not possible to specify how much time is consumed by processes and actions in a system. Also, the tokens used in Petri nets are simple markers which can only be used to indicate whether a place is occupied or not. If a token were allowed to take on values, or better yet, a vector of values, then it could be differentiated from other tokens. Such a token could represent a job and its vectored requests for and uses of modeled systems resources as the token moved about in the net.

Timed Petri nets (Ref 22) are an extension of Petri nets which assign finite firing times to the transitions in a net model. Timed Petri nets can be used to determine the computation rates of activities in a modeled system.

Colored Petri nets (Ref 34) do not assign finite firing times to transition, but they do provide a means of resolving conflicts related to priority of tasks. Tokens are assigned colors which can be used to reflect token priorities when there arises a conflict of several different typed (colors) of tokens in an input place of a transition and one of these tokens must be selected to enable the transition.

Evaluation nets (Ref 15) incorporate both finite

transition firing times and tokens with vectors of values. Evaluation nets (E-nets) were developed specifically for performance evaluation studies, providing "a medium for graphically representing the structure of systems composed of asynchronous events" (Ref 15:IC). In order to facilitate net analysis, a special type of place was introduced, termed a resolution location, which can be used to resolve conflicting situations that may arise in the net (Ref 15:23). These "resolution locations provide a deterministic method to resolve the conflict whenever it does appear during a net operation" (Ref 15:24) which is not available in the Petri net or Timed Petri net models. For a general description of Evaluation nets the reader may refer to Chapter II of this report. For a detailed description and discussion of the development of E-nets and their applications, refer to Reference 15.

Problem Statement

It is desired at the Air Force Avionics Lab at Wright-Patterson AFB to have the ability to perform a high level analysis of proposed avionics computer systems without having to operate a real-time system (as will be done with the DAIS). The aforementioned advantages of model analysis apply when a model is compared to a real-time hardware simulation. The purpose of this investigation is to construct a graph model of the DAIS, a distributed processor avionics system, based on the class of parallel net models known as Evaluation nets. After the model is constructed, it will be evaluated as a tool

for analyzing the performance of a two processor DAIS configuration.

Scope

The intent of this investigation is to use a particular class of graph models to model a particular distributed processor avionics computer system configuration in order to analyze the system's performance. Evaluation nets were selected because they provide the capabilities (conflict resolution, transition execution times, and vectored, valued tokens) to perform a realistic representation of system operation and characteristics (concurrent, asynchronous processing). Additionally, E-nets provide mechanisms for model-environment communication (see Chapter II) which facilitate net model execution. Since the distributed processor avionics systems which the DAIS is intended to simulate do not exist yet, the DAIS itself will be modeled. Analysis of the model can reveal any bottlenecks in the system and provide statistical data on task assignment and execution.

Constraints and Assumptions

Constraints and assumptions are discussed together since they are closely connected. Probably the biggest constraint on the modeling effort was the unavailability of detailed documentation on the Master Executive. The Master Executive is the part of the DAIS operating system responsible for overall system control and data bus communications control. Many

assumptions concerning Master Executive functions were made which cannot be verified or faulted until the said documentation is available (Ref 28.29). These assumptions are discussed in Chapter III, where they can be discussed in context with their impact on the derived model.

Approach

The investigation proceeded in a top-down manner. Prior to constructing any E-net graphs, a set of guidelines were established. As of the writing of this report, only one configuration has been simulated at the Avionics Lab, that being a two-processor configuration of an attack aircraft avionics computer system (Ref 19:24). This was the configuration chosen for the E-net modeling effort.

Three E-net graph models were constructed which represented the functional relationship of the two processors, their bus interface units, and the data bus. The three models represent successively greater levels of detail of the DAIS. The third model represents the DAIS at a task level, and was evaluated as a tool for analyzing the DAIS.

Organization

Chapter II contains a general description of the DAIS and Evaluation-nets. In Chapter III the E-net models are presented with a discussion of the model construction effort. A discussion of the analysis effort is presented in Chapter IV. Results of the investigation and some concluding remarks evaluating this undertaking are presented in Chapter V. Finally, some recommendations for future work are made.

II Background

A general description of the architecture and theory of operation of the DAIS and an informal description of E-nets is presented in this chapter. Enough detail on the DAIS is presented in this chapter to give the reader a basic understanding of the DAIS structure and operation; more detailed discussions of some portions of the architecture and operation are presented in Chapter III to explain the configuration of some parts of the E-net graphs presented there. Only a definition of E-nets is presented in this chapter. For a formal discussion of the development and underlying theory of E-nets and some examples of their application to problems in the field of computer science, the reader is referred to Reference 15.

DAIS

Generally, an avionics system is a collection of separate, and usually dissimilar, specialized systems which communicate through specialized interfaces. Each major avionics function (e.g. navigation, communications, weapon delivery, flight control, and stores management) is supported by its own specialized computer system. A request for navigation data by the flight control function necessitates a specialized interface to support the data transfer. As can be imagined, there are numerous dissimilar computer systems and interfaces being maintained by the Air Force in support of the various aircraft and missions being flown.

DAIS Concept. "The purpose of the DAIS concept is to reduce proliferation and nonstandardization of aircraft avionics, and permit the Air Force to assume initiative in the specification of standard avionic systems and interfaces for future Air Force system acquisitions. The DAIS concept proposes that the processing, information transfer, control and display functions be common and service all the previously described functional areas on an integrated basis" (Ref 23:5). This total system concept is to be realized by incorporating into the DAIS design such features as standardization, modularization, and redundancy. Table 1 lists the DAIS objectives along with the corresponding design considerations to meet the objectives.

General Description. A DAIS configuration is composed of hardware and software building blocks or core elements. The hardware core elements are the multiplex bus system (Bus Controllers, Remote Terminals, and Data Bus), the processors (with associated memory), and the controls and displays (pilot interface). Software core elements are the Operational Flight Program (OFP) and the Operational Test Program (OTP) (Ref 23:5). Additional avionics system elements (sensors, weapons, etc.) are interfaced to the Data Bus via Remote Terminals or, if they are compatible with the multiplex bus protocol (Ref 31), connected directly to the Data Bus. Figure 2 is a functional diagram of a DAIS configuration. The number of processors and the number and types of avionics controls and displays, weapons, sensors, and communications would be dependent on the type of aircraft and its mission and would be specified by the

Table I Design Considerations Utilized in Meeting DAIS Objectives

DAIS Objectives	Design Considerations
Reduce Unnecessary Development Proliferation	Replication (Core Elements) Standardization
Improve Operational Efficiency and Availability/ Maintainability	Automated Aids Digital Technology Redundancy Utilization On-board Test
Improve Flexibility to Changes	Functional Modularity System Modularity
Maintain Basic Avionics Performance	Proven Technology

(Ref 23:6)

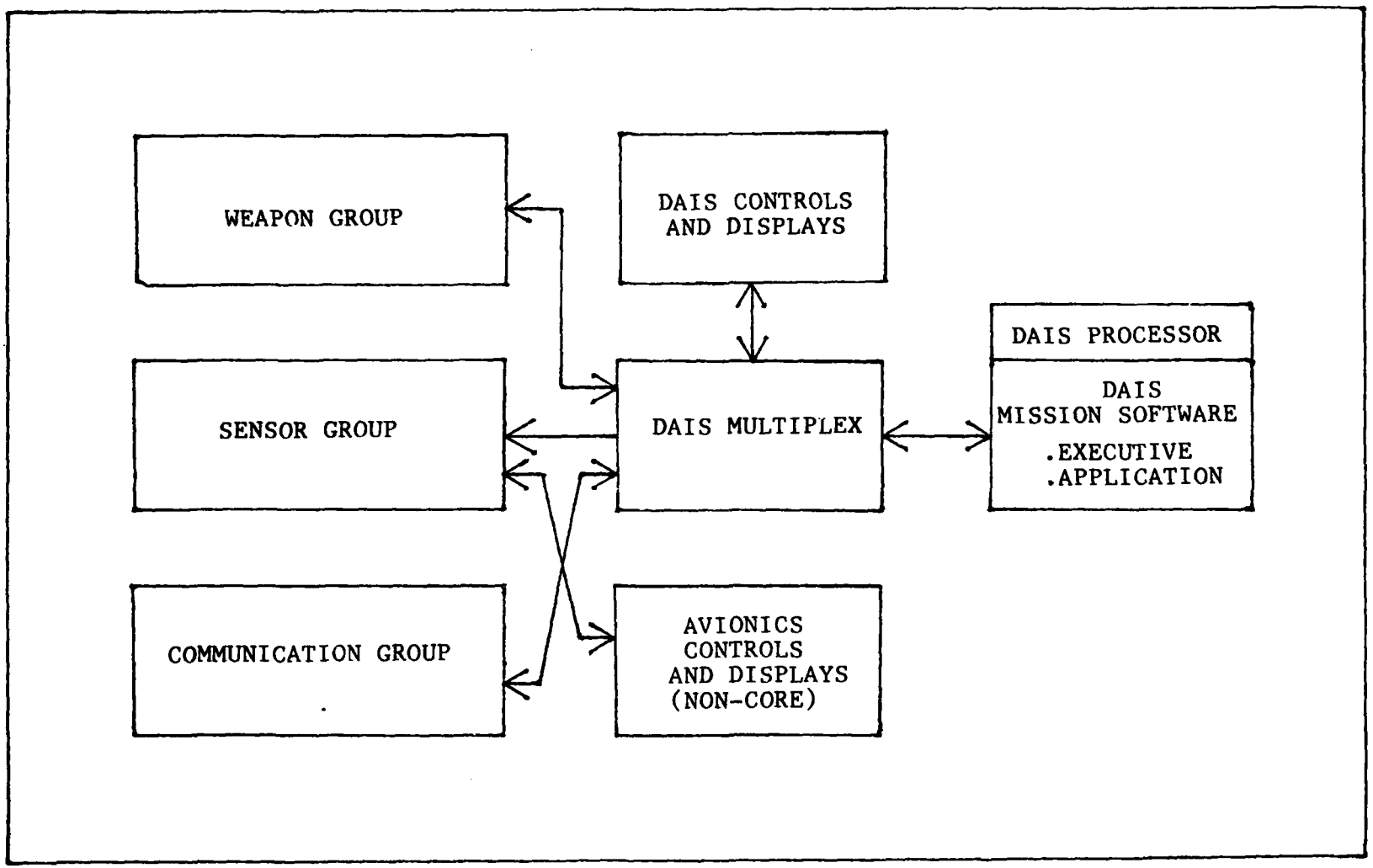


Fig. 2 DAIS Functional Block Diagram (Ref 23:35)

systems designer.

Architecture. The DAIS is structured as a federated network (each processor only has direct access to its own associated memory). A MIL-STD-1553A standardized time division multiplex data bus provides dual redundant communication paths between system core elements (and any bus-compatible avionics elements). The architecture is designed for application to a broad class of configurations where the number of processors can be increased or decreased based on mission processing requirements. Figure 3 shows a general DAIS system architecture.

The DAIS processors are general purpose digital minicomputers specially engineered for airborne use. Operational features include a vectored interrupt priority system, interval timers, floating point arithmetic, and 379K operations per second throughput (based upon a specified benchmark program). Up to 65K words of memory in 16K word modules is available, all of which is directly addressable. Input and output features include discretes, program control, direct memory access (DMA), and external interrupts (Ref 23:15-18).

Associated with each processor in a DAIS configuration is a Bus Control Interface Unit (BCIU). A separate port to memory is provided for the BCIU via a DMA channel. Functionally, the BCIU is an extension of the processor's I/O capability and serves as an interface between the processor and the data bus. Figure 4 illustrates the major components of a BCIU. Timing, control, instruction decoding, and data transfer routing is provided by the Bus Control Module (BCM), a 16-bit

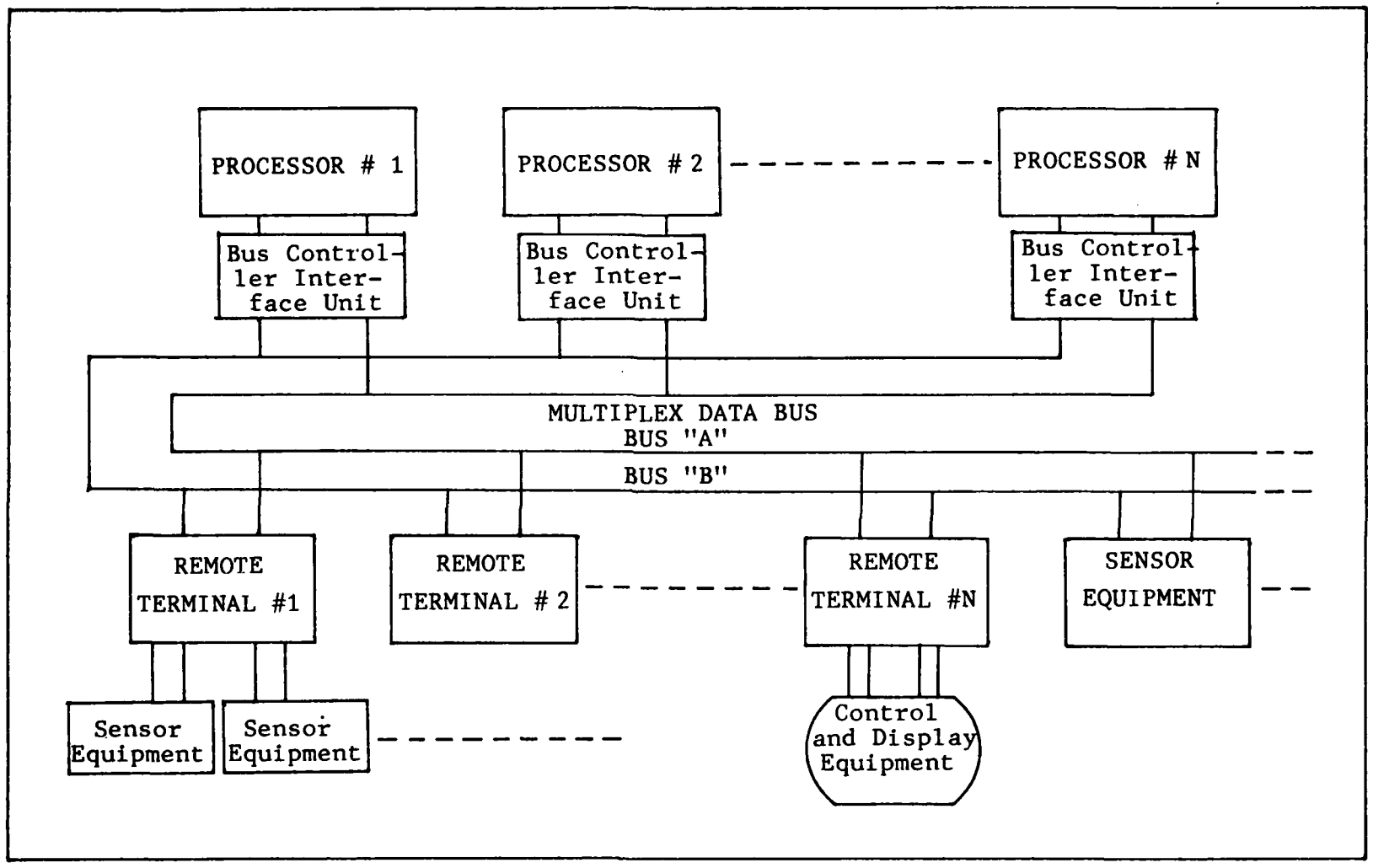


Fig. 3 DAIS Architecture (Ref 23:7)

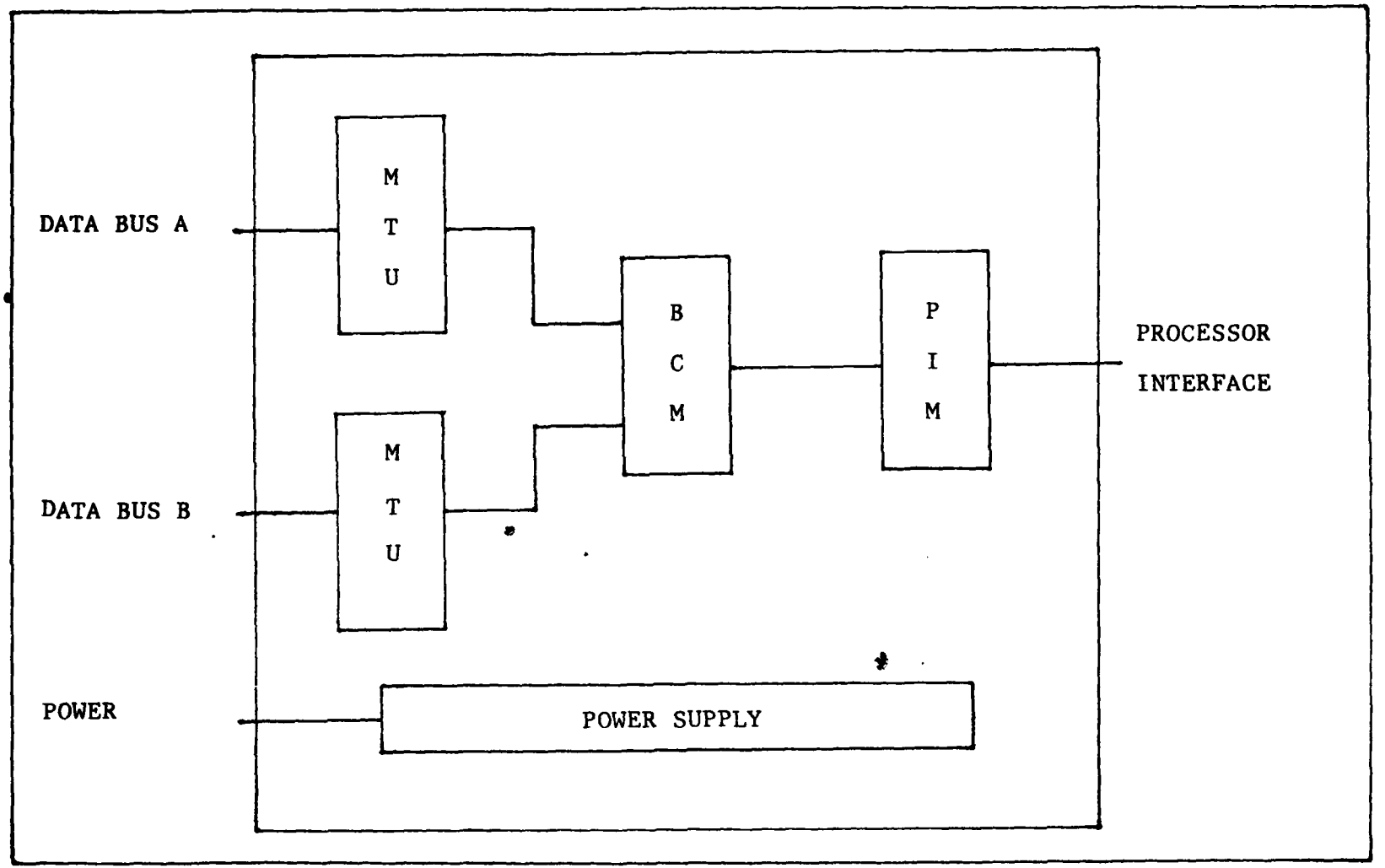


Fig. 4 Bus Control Interface Unit (Ref 23:14)

microprogrammed controller with a 1024 word by 72 bit control memory and a 32-word general register set. Each Multiplex Terminal Unit (MTU) interfaces the BCM to one of the dual redundant data buses, and contains the transmitter and receiver circuits, as well as the logic, to operate the 1.0 mbs data bus. The Processor Interface Module (PIM) contains the input/output interfaces for communications with the associated processor. The PIM has programmed input/output (PIO), direct memory access (DMA), and interrupt capabilities with the processor.

A Remote Terminal (RT) provides the interface between the dual multiplex buses and a subsystem. The RT transfers data in both directions on the Data Bus based on commands received from either bus, and provides status replies in response to commands. The RT partitions messages to the appropriate sub systems it services and conditions the signals to (from) the different subsystems from (to) the Data Bus. The major submodules of an RT can be seen in Figure 5. Each Multiplex Terminal Unit (MTU), interfaces to one data bus, transmitting and receiving information between the bus and a Timing and Control Unit (TCU). The TCU performs all of the timing, control, buffering, decoding, and checking required to transfer information between the MTU and the Interface Modules (IM). Each IM will interface the TCU to a particular subsystem device (e.g. a raster display, aircraft attitude sensor, communication device). A standard set of IM types is available for interfacing the various types of signals possible from

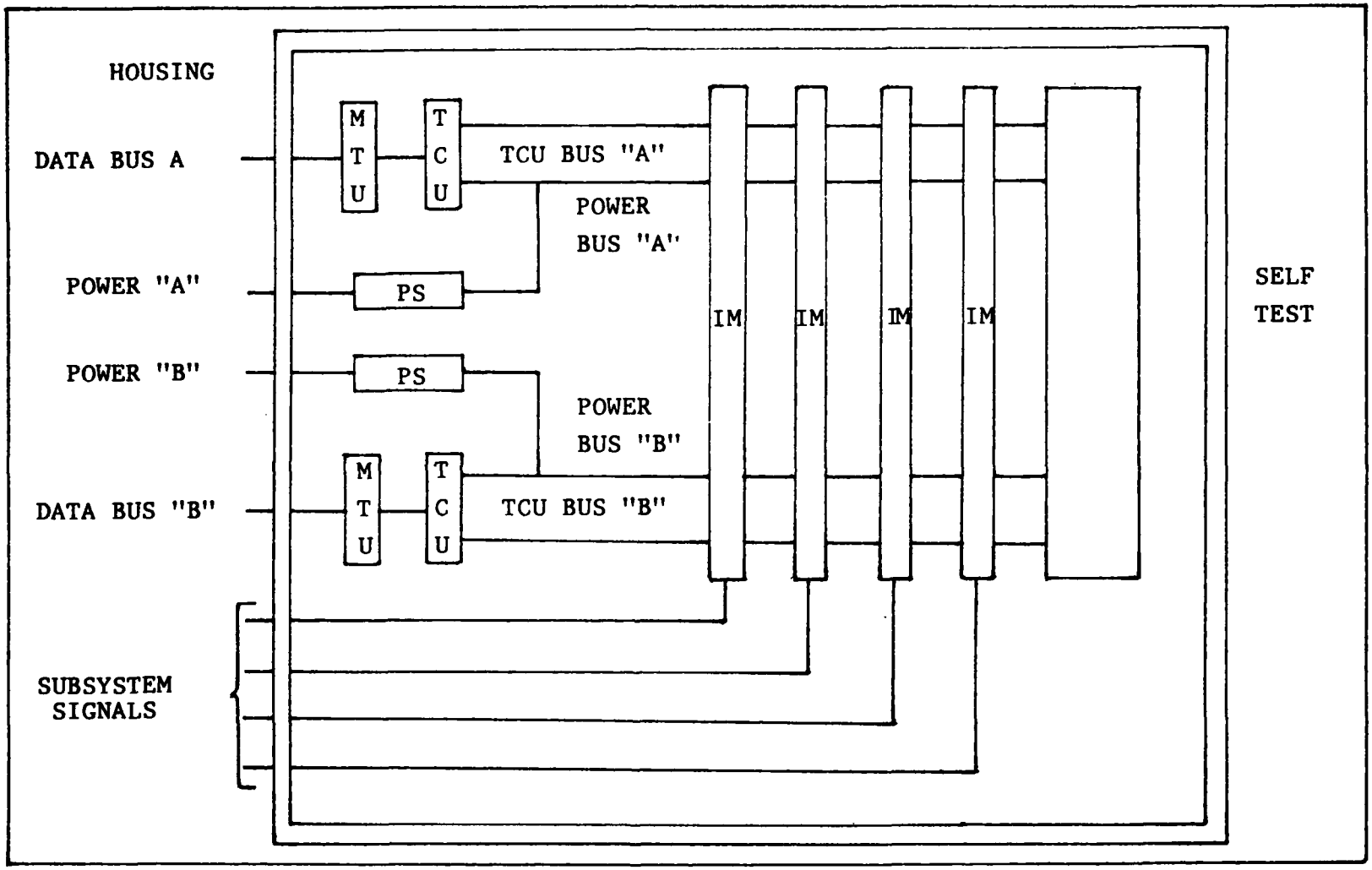


Fig. 5 Remote Terminal Unit (Ref 23:16)

various subsystems. Any type of IM can fit in any IM slot in an RT. The number and types of IMs contained in a particular RT depends on the subsystems serviced by the RT as specified by the systems designer.

Software. The OFP software package is resident within the processors and can be broken down into two major parts: the Executive Software and the Applications Software. The Executive Software serves as the operating system, and is mission and aircraft invariant. The Applications Software changes from mission to mission, from aircraft to aircraft, or as sensors, weapons, and subsystems vary. The OTP contains software for use on the ground (on board the aircraft) to isolate avionics systems failures at the Line Replaceable Unit (LRU) levels. In support of the DAIS objectives, the mission software is characterized by top down structured design, modularization, flexibility to modifications, and portability.

Executive. The Executive Software is further broken down into two functional parts: the Master Executive and the Local Executive. The Master Executive is responsible for overall system control and resides in the processor designated the Master Processor. The Master Processor is given responsibility for control and servicing of the Data Bus. The Master Executive can also reside in a processor designated the Monitor, which provides a back-up to the Master Executive. The Local Executive is resident in the Master Processor and all other processors (termed Remote Processors). The Local

Executive provides real-time services, data input and output, interrupt handling, and task control to the Applications Software. Figure 6 shows a functional breakdown of the DAIS Executive Software. System Initialization and System Control are dependent on the actual system and operating procedures used, and on Master recovery, respectively. Figure 7 shows the physical breakdown of the Executive Software in the processors.

The Master Executive is table driven and manages the system configuration by performing, as a minimum, the following functions (Ref 23:20-21):

1) Data Bus Control - controls transmission of synchronous and asynchronous messages over the multiplexed data bus.

2) System Synchronization Control - allocates time segments (minor cycle and major cycle) for synchronous messages and performs minor cycle synchronization by transmitting master function mode commands to the other processors.

3) System Error Management - monitors and analyzes errors and failures based upon both bus and terminal devices. Initiates message retry procedures to recover from message errors.

4) Configuration Management - detects and isolates permanent device failures, maintains system configuration status, reports failure to the application software, and initiates backup or recovery operation if required.

5) Mass Memory Management - provides for retrieving and storing information from the mass memory on request.

The Local Executive is identical in each processor and is only responsible for those activities in its own CPU. In managing all activities within a processor the Local Executive performs, as a minimum, the following functions (Ref 23:20):

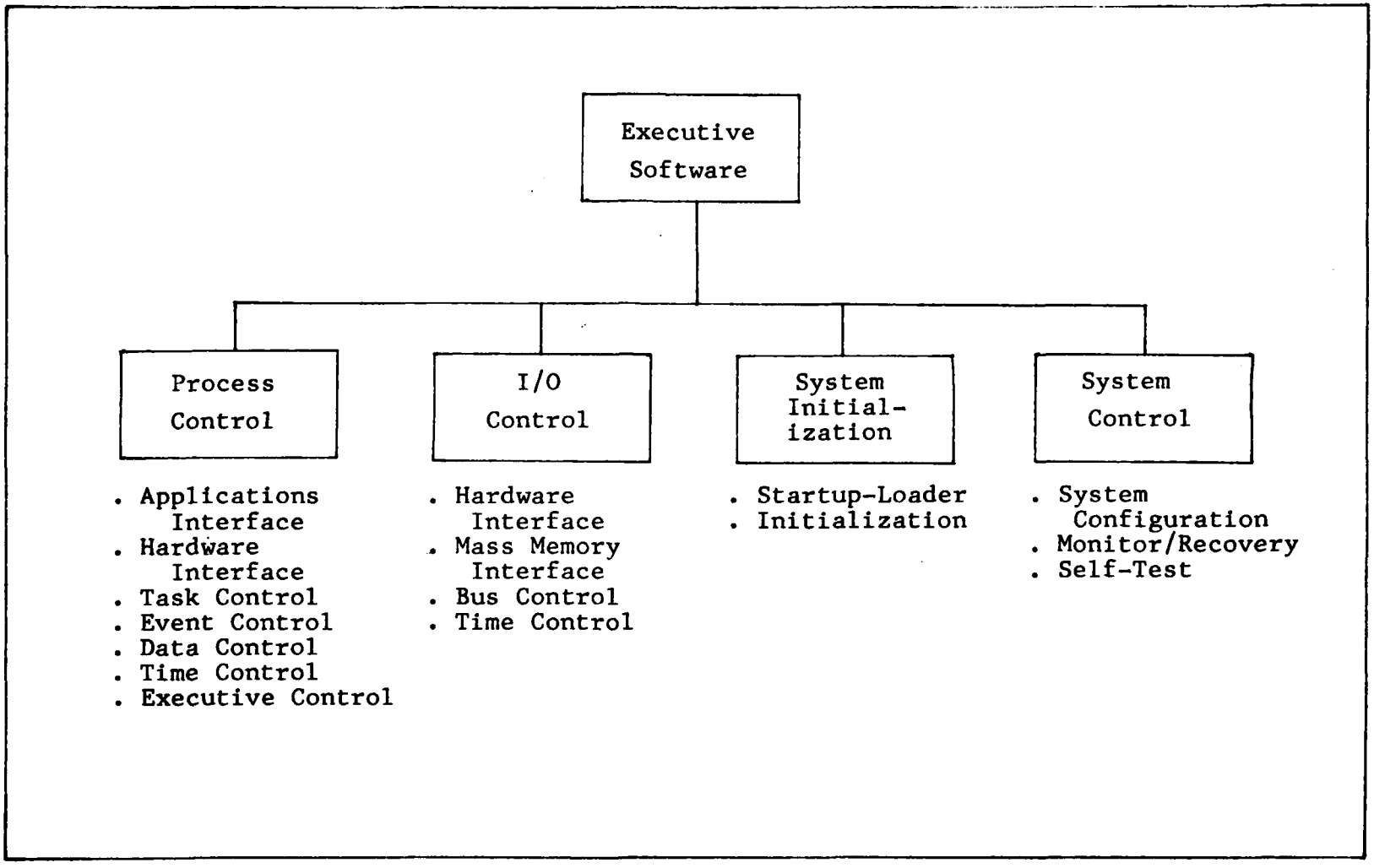


Fig. 6 Executive Functions (Ref 33:6)

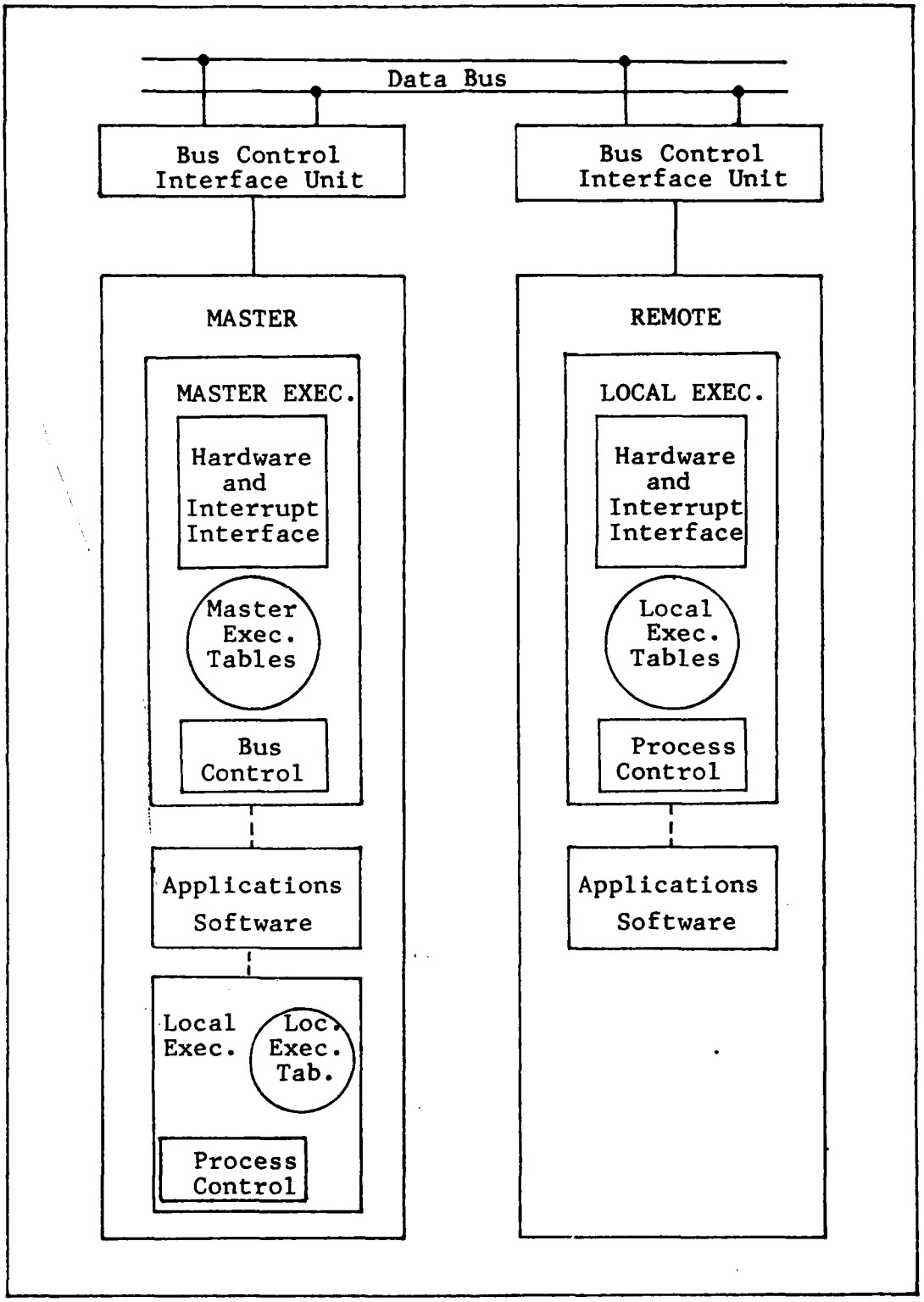


Fig. 7 Executive Physical Breakdown (Ref 33:7)

1) Process Control - provides services for the Applications Software to activate and deactivate periodic and non-periodic tasks when appropriate conditions have been met. The conditions shall be based upon logical setting (on or off) of real time events.

2) Event Control - provides the mechanism for setting, resetting, and evaluating real time events which communicate conditions (on or off) signaled between tasks whether in the same or different processors.

3) Data Control - guarantees integrity of shared data and provides mechanism for transmission and reception of data over the multiplex bus.

4) Processor Initialization - provides initialization for processor power transient recovery, initial program load, and minor cycle synchronization with the other processors.

Applications Software. The Applications Software can be broken down into more basic blocks: Compool blocks, Tasks, Comsubs, and Events (Ref 33:3). Compool blocks contain the global data and are centrally defined and controlled. Tasks are the real-time processes within the system. Comsubs are commonly used subroutines that perform calculations only, have no real-time control or interaction, and communicate solely through parameter passage (no access to Compool blocks). Events are units of binary valued information that enable tasks and the environment to interact (Ref 33:3).

The DAIS Executive is priority driven. The Task with the highest priority which is capable of execution obtains the processor. A given Task must declare the Tasks to which it refers. the Events it uses, the Comsubs it uses, and any Compool data referred to (Ref 33:3).

The Applications Software implements the specific mission avionics functions (e.g. navigation, weapon control, flight

plan, etc.). The Applications Software consists of tasks to control the various mission dependent sensors (e.g. navigation aids, communications, radar and subsystems (e.g. weapons), and perform the various mission functions mentioned above. The Applications software is organized in a heirarchical control tree structure as seen in Figure 8. All Applications Software functions are either controllers or calculators. Each calculator is controlled by a unique controller (with the exception of common service routines). A module (as represented by a node in the tree in Figure 8) can only invoke modules in its own level or itself. However, the events on which activation of a module is based can be signalled by modules at other levels (Ref 20:22).

The functional categories of task modules in the Applications Software are seen in Figure 9 and specified as follows (Ref 18:10-12):

1) The Master Sequencer - is at the top of the Applications Software hierarchy and controls the request processor, configurator, and subsystem status monitor. The Master Sequencer is only required in the Master and Monitor processors.

2) The Request Processor - responds to pilot's requests and invokes the appropriate application functions to generate information for displays or to control avionic support subsystems.

3) The Configurator -
    a) Defines the set of application functions to be invoked by request from the Request Processor or the Subsystem Status Monitor.
    b) Enables a new set of available functions upon significant changes in mission phasing or equipment moding. This set of functions is based on a global knowledge of overall system status.

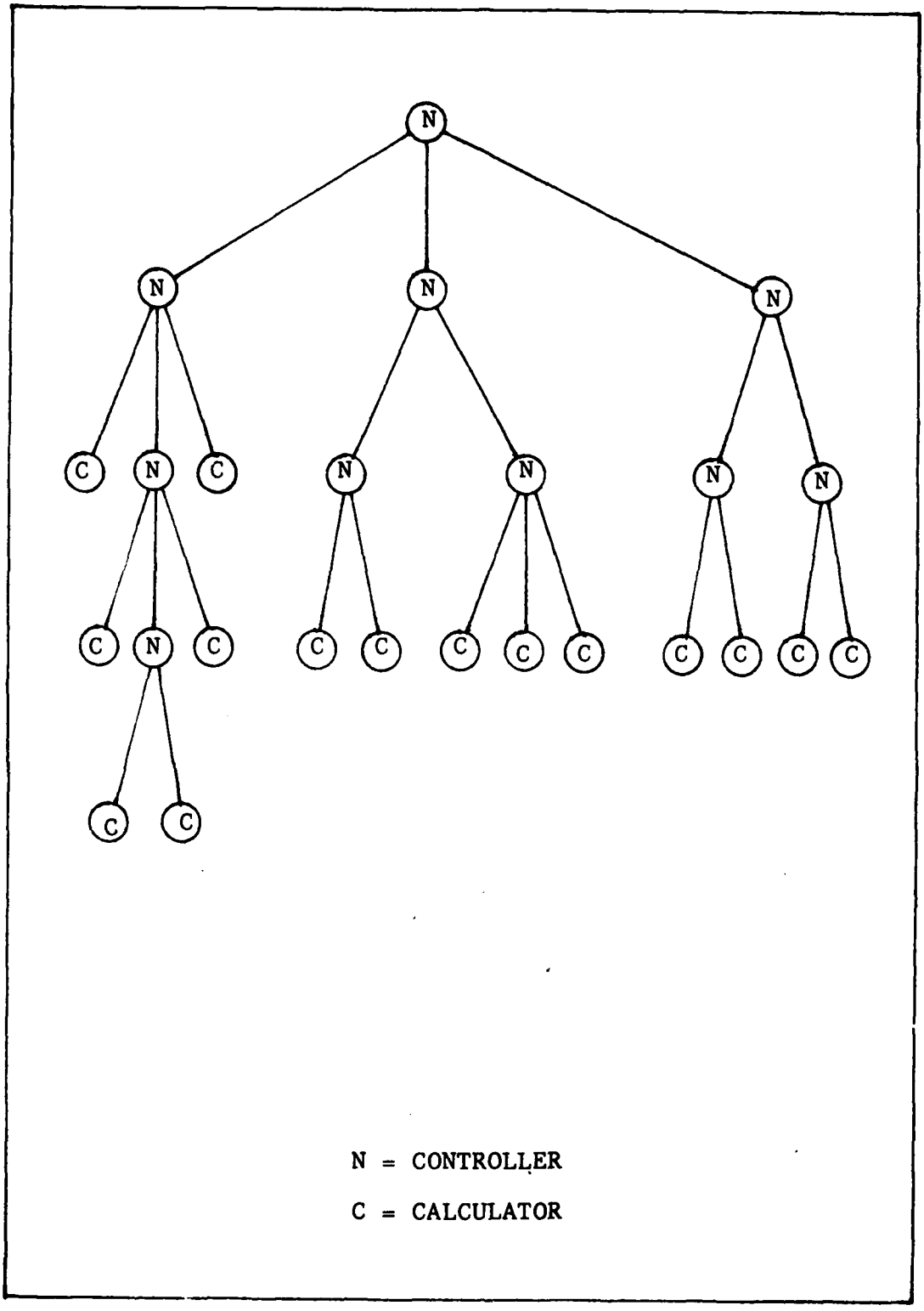


Fig. 8 Hierarchical Control Structure (Ref 20:23)

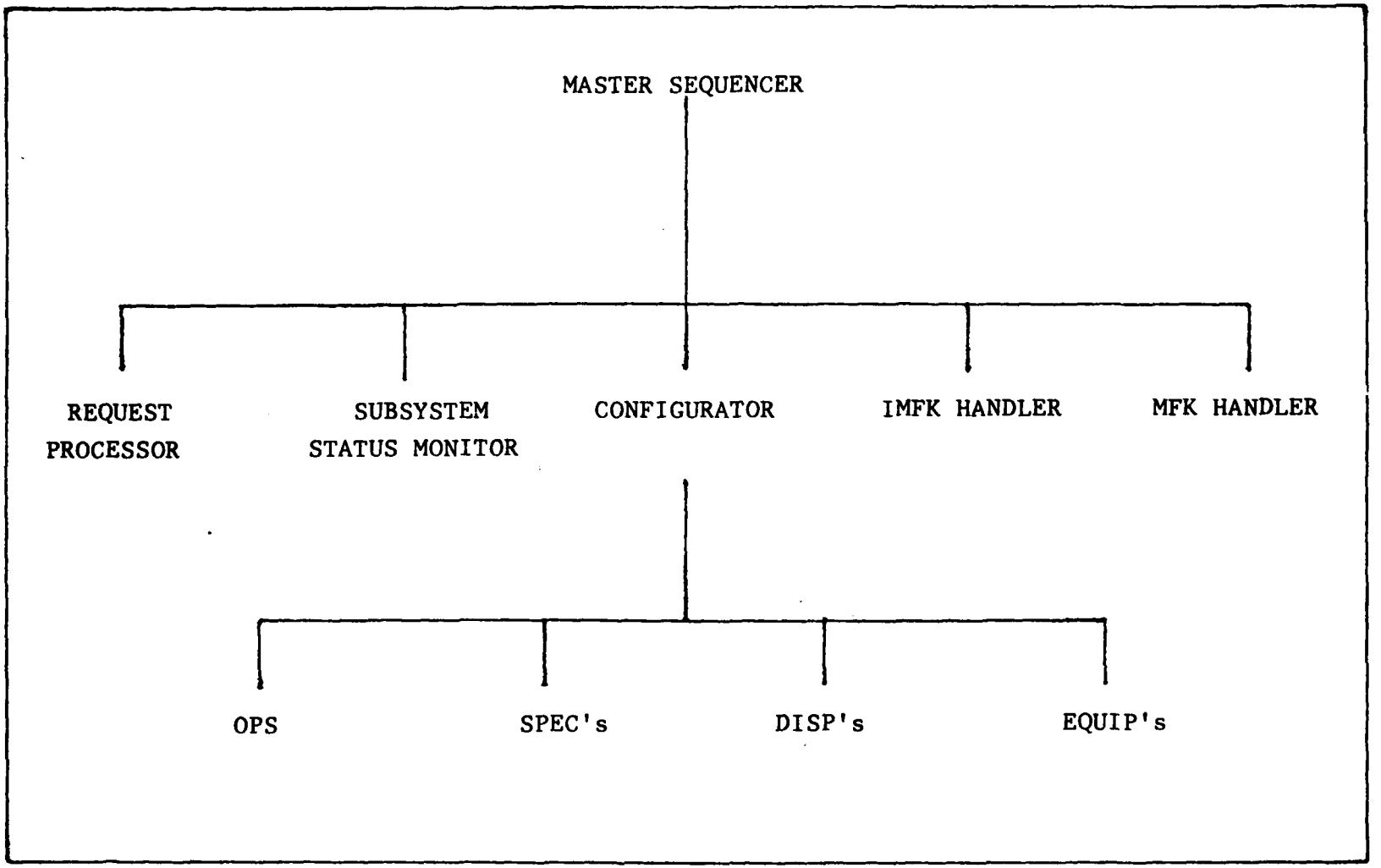


Fig. 9 Application Software Tree (Ref 23:98)

4) The Subsystem Status Monitor - monitors subsystem failures and communicates the information to the Configurator.

5) Mission Operations (OPS):

   a) Act as a control authority for operations either when invoked by phase selection by the pilot or automatically when current phases are terminated in an emergency or abnormal condition.

   b) Exist, as a minimum, for preflight/inflight startup, take-off, cruise, weapon delivery, approach landing, and postflight shutdown operations.

6) Specialist Functions (SPECS) - provide functions required by OPS modules or by the pilot (e.g. navigation, weapon test, etc.).

7) Equipment Handlers (EQUIPs) - exist for each piece of equipment (e.g. communication, radar, etc.), translating the data formats as required for each equipment and managing the control mode of the equipment.

8) Display Processors (DISPs) - provide data interface and control functions in order to manage display presentations, such as messages, symbols, and graphical displays on the DAIS controls and displays.

9) The Integrated Multifunction Keyboard (IMFK) Handler- receives and interprets the pilot-selected IMFK keyed information.

10) The Multifunction Keyboard (MFK) Handler - is similar to the IMFK Handler and acts as a backup in the event of a IMFK failure.

Normal Operation. The DAIS is a real-time system in which the Applications Software processes are coordinated with the passage of real-time in the DAIS environment. Timing within the DAIS is specified in terms of minor cycles and major frames. A minor cycle is the time required to execute the shortest synchronous action and a major frame is the longest interval of a

synchronous action by the Executive. The number of minor cycles per major frame is determined by the applications programmer and is fixed upon system initialization. It is possible to specify the time of an action within one minor cycle. I/O interactions, interprocessor interactions, and task interactions may occur, may be known, and may be controlled within the framework of the minor cycle time granularity (Ref 25:5-6). Minor cycle synchronization is used to coordinate the actions of all processors.

The beginning of a minor cycle is signaled by a timer interrupt (time A interrupt) in the Master Processor. If the current minor cycle's synchronous bus message list has been transmitted, then the Master Executive starts the minor cycle bus message list for the new minor cycle. The minor cycle bus list is a bus message list of minor cycle interrupts to each Local Executive (Ref 9:51-53). Upon receipt of the interrupt, the Local Executive calls the Minor Cycle Setup Routine which performs several functions: a) schedules Tasks to be executed during the new minor cycle, b) sets DMA pointers to the appropriate message receive and transmit blocks, and c) in the Master Processor, the Master Executive is called to start processing the synchronous bus message list for the new minor cycle (Ref 26:144-156).

Bus Control. Bus control operations form the heart of the system (Ref 9:40). The Master Processor and its BCIU control bus protocol and data flow on the bus.

An example of synchronous messages is the transfer of inertial navigation system output data from a Remote Terminal to a navigation applications function in a processor. Synchronous message operations attend to the predetermined, scheduled flow of data. A synchronous message is assigned a period and phase; the period is the number of minor cycles between message transmissions, and the phase is the displacement, in minor cycles, relative to the first minor cycle (Ref 9:51). Asynchronous message operations manage the data flow which cannot be scheduled in advance. An example of an asynchronous message is a request from the pilot, via a Remote Terminal, for a readout of the distance to target.

Synchronous bus messages are controlled by a predefined bus instruction list which is part of the Master Executive's preloaded tables. Each minor cycle, the particular subset of synchronous messages to be transmitted during that cycle are linked to the Master BCIU's list of pending bus messages. Asynchronous messages, which are also predefined, are scheduled by the Master Executive in response to requests from Remote Terminals and applications tasks within the processors. In general, asynchronous message operations are given priority over synchronous operations (Ref 9:42).

All Remote BCIUs (in conjuction with their connected processors), Remote Terminals, and bus-compatible avionics subsystems are considered as Remote Devices. Every bus operation is initiated by the Master BCIU under control of the Master Processor. The Master BCIU places a command on the bus

lines and then monitors the bus for a status response by the Remote Device. A Remote Device monitors the bus for commands directed to it. Upon detecting a command addressed to itself, the Remote Device responds by performing the operation and then placing a status word on the bus lines. By setting selective bits in the status word, the Remote Device can request services from the Master Processor/BCIU or inform the Master Executive of a detected failure.

Some bus operations, such as transmission and reception of synchronous messages, require no intervention by the processor. The BCIU uses DMA to transfer the data to (from) the processor from (to) the bus. Asnychronous bus operations are among those bus operations which require the BCIU to interrupt its processor in order that the Local Executive can process the interrupt and take appropriate action.

Task Control. Tasks and Comsubs are the processing modules of the Applications Software. Real-Time Statements and Real-Time Built-In Functions are used by Tasks to invoke Local Executive services to control and reference the state of other Tasks and the values of Events and Compool Block data (Ref 25:9). To understand how Tasks interact, it is necessary to understand the possible states a Task may be in. Each Applications Software Task is in one of the states shown in Figure 10 at any given instant in real time. Not all states are mutually exclusive. A SUSPENDED task is also INVOKED, ACTIVE, and DISPATCHABLE. The Real-Time Statements SCHEDULE, WAIT, CANCEL, and TERMINATE are used by Tasks to change the

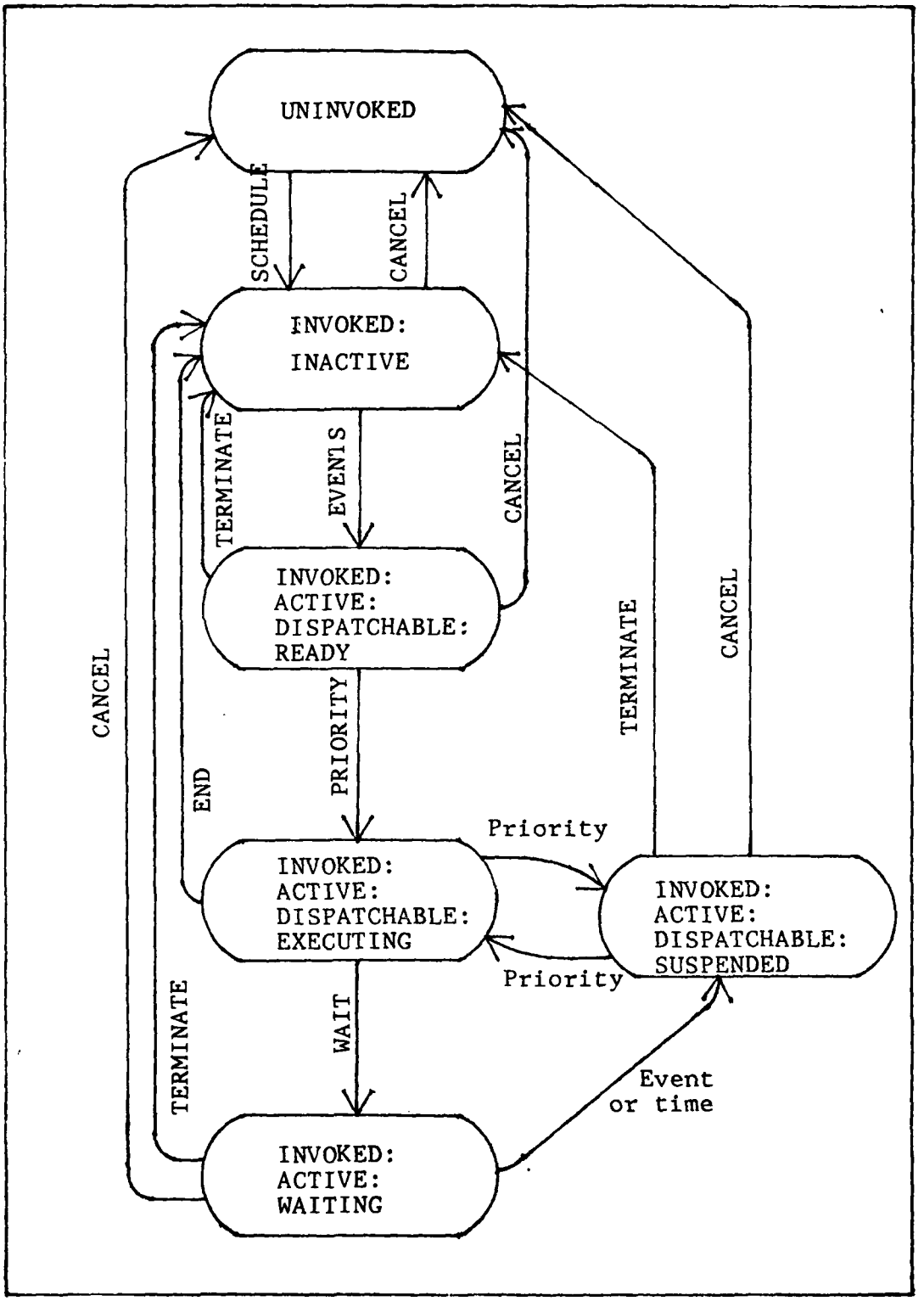


Fig. 10 Task State Transition Diagram (Ref 25:10)

states of other Tasks (or themselves in some cases).

Associated with each Task is an Event Condition Set. The Event Condition Set is a set of Conditions, each of which is associated with one or more Events. Dependent on the occurrence of these Events, each Condition has a value of ON or OFF. The "desired value" of the Condition may be either ON or OFF, and when all the Conditions in an Event Condition Set have their respective "desired values", the associated Task is ACTIVATED by the Executive (Ref 25:11-18).

A Task returns from the ACTIVE state to the INACTIVE state for one of two reasons: a) it completes execution or b) another task forcibly TERMINATES it. In either case, immediately after being placed in the INVOKED/INACTIVE state, the Task's Event Condition Set is evaluated, and if all Conditions have their desired values, the Task is immediately ACTIVATED (Ref 25:11).

A Task that is ACTIVATED is immediately put to the DISPATCHABLE state. All DISPATCHABLE Tasks are capable of being executed. Whenever the Executive passes control to the Applications Software, the DISPATCHABLE Task with the highest priority is selected and executed. Once a Task is in the ACTIVE/DISPATCHABLE/EXECUTING state, one of three things can happen to it: a) it completes execution and goes to the INACTIVE state (see above), b) a higher priority Task becomes DISPATCHABLE, and the currently executing Task is SUSPENDED, or c) the Task executes a WAIT statement (which specifies a desired value for an Event or a future time), which causes

the Executive to place the Task in the WAITING state. When the waited-for condition is satisfied, a WAITING Task is again made DISPATCHABLE. A WAITING or SUSPENDED Task may be CANCELED or TERMINATED by another Task.

At any time, there may be many processes potentially executable within a processor (Ref 25:12). These processes include Tasks, and Executive actions invoked by Tasks in other processors or in response to an RT-generated request. A system of priorities exists to resolve these conflicting demands for the processor. There are two classes of Tasks: Normal Mode Tasks and Privileged Mode Tasks. As a group, Privileged Mode Tasks and Executive actions have higher priority than Normal Mode Tasks (Ref 25:12).

Once a Privileged Mode Task or an Executive process starts executing, it runs to completion. If an Executive action is invoked by an executing Privileged Mode Task or Executive Process, the invoked process is executed as if it were an inline block of code. A Privileged Mode Task or an Executive process cannot be interrupted by a Task or Executive action requested or invoked outside of itself (Ref 25:12). When no Privileged Mode Task is ACTIVE and no Executive action is pending, the highest priority DISPATCHABLE Normal Mode Task is executed.

Theoretically, Normal Mode Tasks are linearly ordered by priority while Privileged Mode Tasks are executed on a First-come-first-serve basis. In actuality, there exists within each processor a table of entries, one for each Task, for all

of the Applications Tasks (Normal and Privileged Modes) residing within the processor. This table, named Task Table B, is ordered by priority, with Privileged Mode Tasks first, and the lowest priority Normal Mode Task last (Ref 26:21). The relative order of the Privileged Mode Tasks is arbitrary, but fixed at compilation time.

Immediately following system initialization, one Task, the Master Sequencer, is INVOKED by the Executive, while all other Tasks remain in the UNINVOKED state (Ref 25:11). Thereafter, any Task may be INVOKED by a schedule statement or UNINVOKED by a CANCEL statement executed by another TASK.

Summary. The Digital Avionics Information System (DAIS) is a distributed processor avionics computer system. The major hardware elements are the multiplex bus system (Bus Controllers, Remote Terminals, and Data Bus), the processors (with associated memory), and the controls and displays (pilot interface). Additional avionics system elements (sensors, weapons, etc.) are interfaced to the DAIS system through Remote Terminals attached to the Data Bus or through direct attachment to the bus. The major software elements of the DAIS are the Operational Flight Program (OFP) and the Operational Test Program (OTP). The OFP has two major parts, the Executive and Applications Software. The Executive serves as the operating system, providing overall system control, bus control, and real-time services to the Applications Software. The Applications Software implements the avionics functions (e.g. navigation, weapon control, flight plan, etc.), and can be thought of as

the user tasks serviced by the Executive. The Applications Software is structured as a hierarchical control tree with tasks at the nodes. The topmost task in the hierarchy, the Master Sequencer, is scheduled immediately following system initialization. Thereafter, tasks are executed after they are scheduled by the task immediately above them (and, of course, in line) in the control tree structure (see Figure 8). The OPT is composed of software used on the ground to isolate avionics system failures.

Evaluation Nets

The remainder of this chapter is devoted to the definition of Evaluation nets (E-nets). The axioms and definitions presented are from the formal definition of E-nets in Reference 15. Several axioms and definitions are presented which lead to the formal E-net definition presented near the end of this section. An example of the use of E-nets to model a simple computer system is provided and will be referred to repeatedly. First, a basic definition of E-nets is developed. Next, the concept of "net flow" is discussed and defined. Then, the formal E-net definition is developed and presented. Lastly, the concept of "macro structures" is introduced and an example of a macro structure is presented.

Basic Definition: E-nets are characterized as marked interpreted directed graphs (Ref 15:Ib). An E-net is composed of two sets of nodes, transitions and locations, interconnected by "directed" arcs. Transitions are only connected to locations and locations are only connected to transitions via arcs directed into or out of the nodes. In Figure 11, location b_1 is an input location to transition a_1 and location b_2 is an output location of a_1, as evidenced by the directed arcs connecting these nodes.

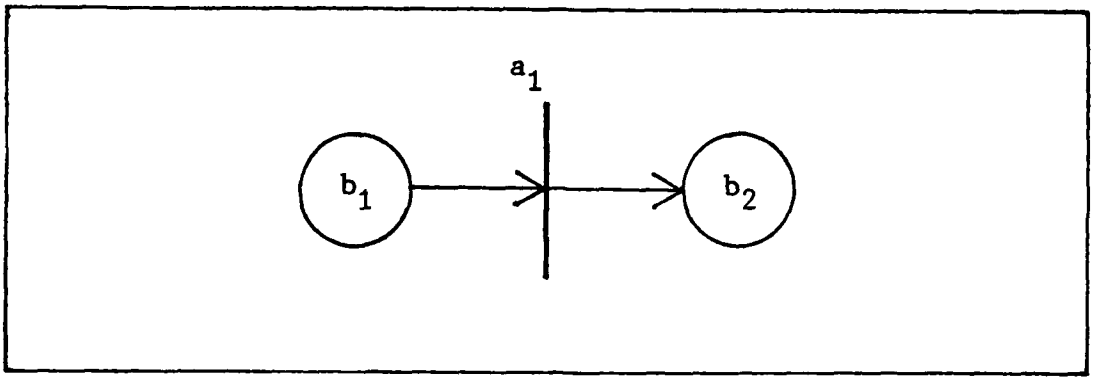


Fig. 11 Simple E-Net

The structure of a parallel system can be pictorally represented by an E-net graph. The transitions represent events or actions in the system and the locations represent conditions which exist in the system. A transition action causes the status of the locations (and, thus, the conditions they represent) attached to the transition to be altered according to the particular transition definition (Ref 15:30). A token is a marker which may reside on a location and indicated the status of the location (Ref 15:30). The placement of a token on a location represents a corresponding condition which is satisfied. An E-net with tokens is a "marked", directed graph.

By assigning a set of tokens to a subset of the locations in an E-net graph, and then executing the transition actions as the transitions are enabled (in accordance with the rules and definitions which follow), the token flow through the E-net can represent the dynamic activity of the modeled system. When an enabled transition fires, it removes tokens from a subset of its input locations and places tokens on a subset of

its output locations (Ref 15:Ib). The transition firing time specification and the modification of the status of the attached locations provide an interpretation of the transition function (Ref 15:74), completing the characterization of E-nets as marked, "interpreted", directed graphs.

Basic Definition. This section presents the axioms and definitions that underlie the basic definition of an E-net. For a thorough discussion of the theory and development of E-nets, the reader is referred to chapter 2 of Reference 15.

Axiom 1 (Ref 15:30): The maximum number of locations connected to a transition is four. The limitation facilitates precise definition of the concept of transition action (Ref 15:30): the number four is linked to the number of primitive transition types (5) provided in E-nets. A location may be an output location for, at most, one transition and/or an input location for, at most, one transition. A direct result of this restriction is that two or more transitions will not compete for a token on a common input location (race condition).

Definition 1 (Ref 15:30): "A location is empty if it does not contain a token and full if it contains a token. If it is not known whether the location is empty or full, the status of the location is undefined."

Axiom 2 (Ref 15:30): "A location may change from empty to full or full to empty only by the action of one of the transitions to which it is connected."

Axiom 3 (Ref 15:31): "The action of a transition is defined by the mapping whose domain and range are the status

of locations connected to the transition." For representation purposes, the status of locations attached to a transition are represented by an n-tuple, where n is the number of locations attached to the transition. The values that a coordinate in the n-tuple may take are: a) 1 if the status is full, b) 0 if the status is empty, and c) ϕ if the status is undefined.

Definition 2 (Ref 15:32): "A peripheral location is a location having exactly one connection to one transition. Locations oriented into (out of) a transition are input (output) locations of the transition. All locations that are not peripheral locations are inner locations."

Definition 3 (Ref 15:32): A resolution location (r-location) is a location that is directed into an X or Y transition (see Definition 4). Based on the r-location status of 0 or 1, an alternate output location of an X transition or an alternate input location of a Y transition is selected for the action of the transition. A peripheral r-location may take on the values 0,1, or ϕ. An inner r-location may only take on the values 0 or 1 (Ref 15:32).

Axiom 4 (Ref 15:33): "Only resolution locations may have an undefined status."

Definition 4 (Ref 15:35): The following five basic transition types are provided in E-nets. Letting b_1 and b_2 represent input locations, b_3 and b_4 represent output locations, and r represent a resolution location, the basic transition types with their corresponding mappings are:

$T(b_1,b_3)$:	$(1,0) \rightarrow (0,1)$
$J(b_1,b_2,b_3)$:	$(1,1,0) \rightarrow (0,0,1)$
$F(b_1,b_2,b_3)$:	$(1,0,0) \rightarrow (0,1,1)$
$X(r,b_1,b_3,b_4)$:	$(0,1,0,0) \rightarrow (e,0,1,0)$
	$(0,1,0,1) \rightarrow (e,0,1,1)$
	$(1,1,0,0) \rightarrow (e,0,0,1)$
	$(1,1,1,0) \rightarrow (e,0,1,1)$
$Y(r,b_1,b_2,b_3)$:	$(0,1,1,0) \rightarrow (e,0,1,1)$
	$(0,1,0,0) \rightarrow (e,0,0,1)$
	$(0,0,1,0) \rightarrow (e,0,0,1)$
	$(1,1,1,0) \rightarrow (e,1,0,1)$
	$(1,1,0,0) \rightarrow (e,0,0,1)$
	$(1,0,1,0) \rightarrow (e,0,0,1)$

"If r is a peripheral location, replace the character, e, by ϕ: otherwise replace e by 0" (Ref 15:35). An X or Y transition action leaves the status of peripheral r-locations undefined (how their status becomes redefined is discussed later). Figure 12 illustrates the graph representations of the corresponding above transition types. The vertical bars in the graphs represent the transitions, the hexagons represent resolution locations, and the circles represent standard locations. The mappings of the left-hand tuples into the right-hand tuples in Definition 4 correspond to the "firings" of the given transitions. A more formal definition of a transition firing is available.

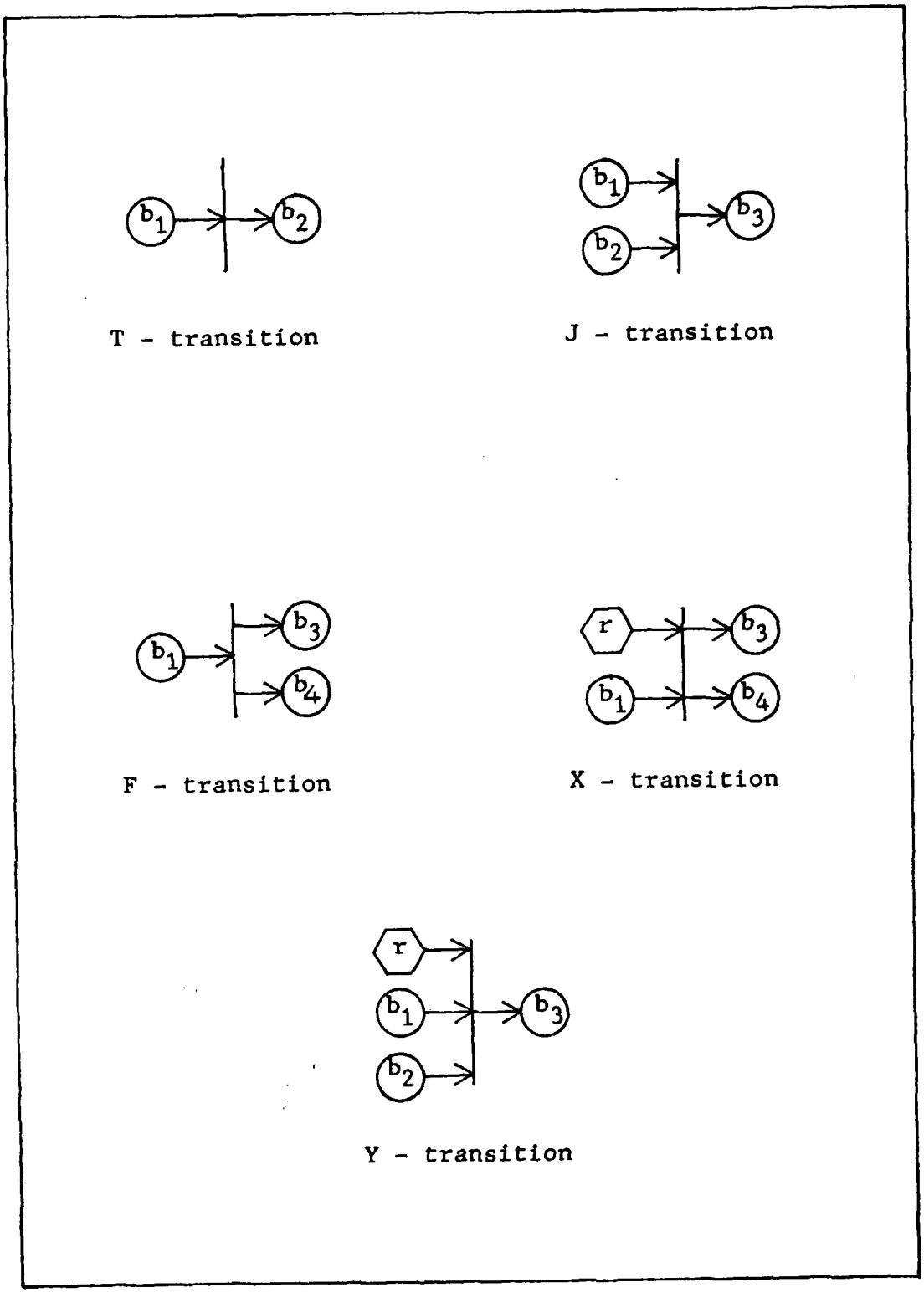


Fig. 12 Basic Transition Types (Ref 15:35)

Definition 5 (Ref 15:36): "A transition firing is a three phase operation consisting of the following phases:

1) Enable phase - at the instant the status of the locations connected to the transition satisfy the left-hand side of the transition's definition, the transition is enabled and begins operation.

2) Active phase - The transition action is in progress, but the status of all locations attached to it remain unchanged.

3) Terminate phase - The transition action is completed, instantaneously changing the status of all associated locations to agree with the right-hand side of the transition schema."

Definition 6 (Ref 15:37): A transition with a peripheral resolution location is pseudo enabled whenever the status of the associated locations agrees with the left-hand side of the transition schema except for the resolution location whose status is undefined (ϕ). A pseudo enabled transition can only become enabled when the net environment causes the status of the resolution location to become full or empty.

Definition 7 (Ref 15:38): The transition time (firing time or processing time) of a transition, a, is denoted t(a), and is the time required for the active phase of a firing of transition a. T(a) may be a constant value for all tokens that enable a. For an X or Y transition, t(a) may depend on which alternate path is taken (denoted t_0 (a) if the "0" path is taken and t_1(a) if the "1" path is taken) (Ref 15:39). The enabled and termination phases of a transition firing require zero time. These two phases merely represent the instantaneous setting of values in the domain and range of a transition mapping, respectively.

Definition 8 (Ref 15:40): An Evaluation net structure, E, is denoted by

$E = (L,P,R,A)$ where

L = a finite, non-empty set of locations,

P = a set of peripheral locations, $P \subseteq L$,

R = a set of resolution locations, $R \subseteq L$, and

A = a finite, non-empty set of transitions, $\{a_i\}$, where

$a_i = (s,t(a_i))$; $s \in S$ and $t(a_i) \in T$:

S = a finite, non-empty set of transition types satisfying Definition 4,

$T = \{t(a_i) \mid * (a_i)$ is a transition time for $a_i\}$.

Figure 13 shows an E-net structure whose formal description is as follows (Ref 15:40-41):

$E = (L,P,R,A)$

$R = \{r_1,r_2,r_3,r_4\}$

$P = \{b_1,b_{13}\} \cup R$

$L = \{b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_{10},b_{11},b_{12}\} \cup P$

$A = \{a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8\}$

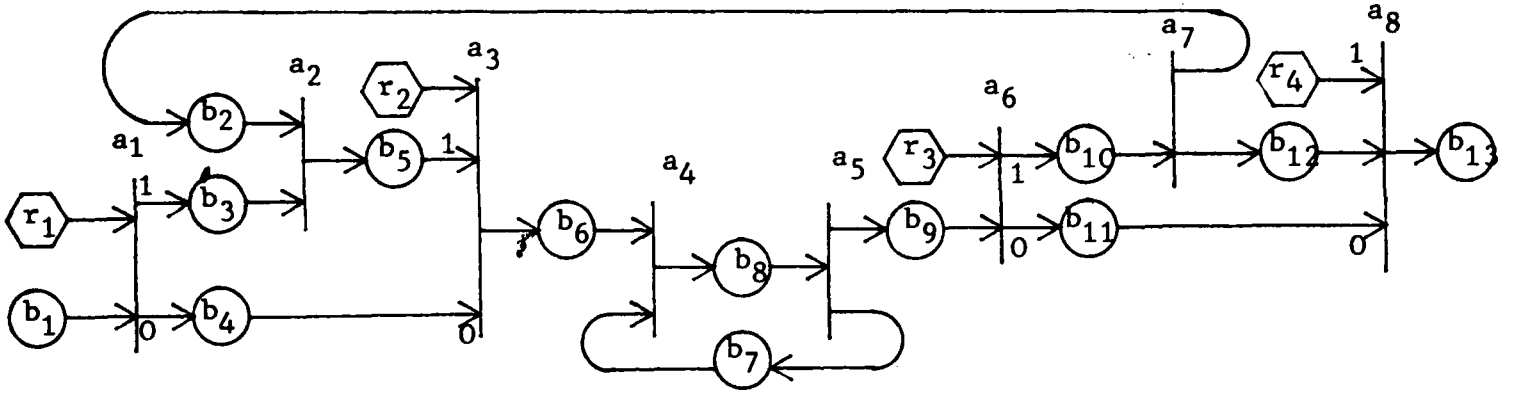
$a_1 = \{X\ (r_1,b_1,b_4,b_3),(t_0(a_1),t_1(a_1))\}$

$a_2 = \{J(b_2,b_3,b_5),t(a_2)\}$

$a_3 = \{Y(r_2,b_4,b_5,b_6),(t_0(a_3),t_1(a_3))\}$

$a_4 = \{J(b_6,b_7,b_8),t(a_4)\}$

$a_5 = \{F(b_8,b_9,b_7),t(a_5)\}$



b_1: Job ready to enter mix
b_2: Tape drive is available
b_3: Job requires tape drive
b_4: Job does not require tape drive
b_5: Tape job has drive allocated
b_6: Job is requesting CP
b_7: Central processor is idle
b_8: Central processor is allocated
b_9: Job is through with the CP
b_{10}: Tape job ready to release drive
b_{11}: Non-tape job ready to vacate
b_{12}: Tape job ready to vacate
b_{13}: Job is complete
r_1 : Route tape jobs to b_3, non-tape to b_4
r_2 : Chooses job from b_4 or b_5
r_3 : Routes tape job to b_{10}, non-tape to b_{11}
r_4 : Chooses job from b_{11} or b_{12}

Fig. 13 E-Net Graph of Simple System (Ref 15:28)

$$a_6 = (X(r_3,b_9,b_{11},b_{10}),(t_0(a_6),t_1(a_6)))$$

$$a_7 = (F(b_{10},b_2,b_{12}),t(a_7))$$

$$a_8 = (Y(r_4,b_{11},b_{12},b_{13}),(t_0(a_8),t_1(a_8)))$$

where the $t(a_i)$ are arbitrary transition time expressions.

Figure 13 is a simple E-net model of a computer system executive where a job which enters the mix may need a tape drive. A job that needs a tape drive must be allocated one (transition a_2 fires) before it can proceed to request the central processor (location b_6). If the processor is idle (b_7 contains a token) and a job requests it (b_6 contains a token) then the processor is allocated (transition a_4 fires). After a job has finished with the processor, the processor is deallocated and, if applicable, then the tape drives are released. It should be noted that the resolution locations, r_1,r_2,r_3, and r_4, serve to avoid possible conflicts in the net. r-locations r_1 and r_3 determine which output location will receive a token when their respective transitions fire. r-locations r_2 and r_4 determine which input location is to be considered the enabling location (and will lose a token) when the respective transitions fire. Since there are no inputs to r_1,r_2,r_3, and r_4 from the net, it is assumed that the environment of the net will set the values of the r-locations (Ref 15:42). It should also be noted that if the transition

times, $t(a_i)$, were given, it would be possible to determine "a time for tokens to traverse the net, yielding the notion of turnaround time," (Ref 15:29).

Net Flow. " The status of an Evaluation net is given by providing the status of each location associated with the net, where each location is empty, full, or undefined" (Ref 15:49). The initial marking of a net necessarily precedes the beginning of operation of the net. Then, either a transition must fire or a peripheral location must become empty or full to change the net marking (Ref 15:50). Additional definitions are provided in the following paragraphs on which the discussion of net flow is formally based.

Definition 9 (Ref 15:50): "Given an evaluation net structure, $E = (L,P,R,A)$, a marking of E is a function, M, taking L into the set $\{0,1,\phi\}$." A marking of a net, E is denoted by

$$M(b_i) = j \quad , \quad j \in \{0,1,\phi\}$$

for all $b_i \in L$. $M(b_i) = 1$ implies that location b_i is full, $M(b_i) = 0$ implies that b_i is empty, and $M(b_i) = \phi$ implies that the status of b_i is undefined. The initial status of L is defined by the initial marking of E , denoted M_0.

An evaluation net can now be defined by the ordered pair (E,MO) (Ref 15:50). Given a net defined by (E,M_0), let a resulting sequence of transition terminations be $a_{i_1}, a_{i_2}, \ldots$. Denoting the status of L (the set of net locations) after

transition a_{i_j} has fired by M_j , the sequence $M_0, M_1, \ldots, M_j$ is the state sequence of the net, E, after j transitions have terminated (Ref 15:50). Referring to Figure 13, let

$$M_0(b_1) = M_0(b_7) = 1 \quad ,$$

$$M_0(b_i) = 0 \quad , \qquad i = 2,\ldots,6,8,\ldots,13 \text{ and}$$

$$M_0(r_j) = 0 \quad , \qquad j = 1,2,3,4 \quad .$$

Then, the state sequence

$$M_0, M_1, M_3, M_4, M_5, M_6, M_8$$

is the state sequence for the net of Figure 13 corresponding to the transition termination sequence

$$a_1, a_3, a_4, a_5, a_6, a_8$$

and with initial marking M_0 .

Definition 10 (Ref 15:51): Given the net $E = (L,P,R,A)$, with $a_1, a_2, \ldots, a_n \in A$ and $b_1, b_2, \ldots, b_n \in L$ where b_i is directed into a_i and b_{i+1} is directed out of a_i, a "path in E of a token K is a sequence of locations that K resides on for ordered firings of $a_1, a_2, \ldots, a_n$ and is denoted by $\gamma_K = b_1, b_2, \ldots, b_n$," (Ref 15:51).

Definition 11 (Ref 15:51): Given (E,M_0), the dwell time of $b_i \in L$, denoted $d(b_i)$, is the total amount of time that the

status of b_i has been full since the net was initialized with M_0. The dwell time of token K on b_i, denoted $d_K(b_i)$, is the total amount of time K resided on location b_i.

It is intuitively obvious that the path of a token in a net may indicate the utilization of facilities represented by the locations. Referring again to Figure 13, locations b_1 and b_{13} represent the input and output control points, respectively, of a computer system executive. The time it takes a token to traverse the path $b_1,\ldots,b_{13}$ in Figure 13 corresponds to the "turnaround time" of the system for a given "task". Likewise, the ratio of the dwell time $d(b_8)$ to the total net operation time corresponds to "central processor utilization". To compute the dwell time of a particular token K on b_8, we use the formula

$$d_K(b_8) = t_j - t_i$$

The value t_i is the system clock time at which the termination phase of a firing of a_4 placed token K on location b_8. Time t_j is the value of the system clock when the termination phase of a subsequent firing of a_5 moves token K from b_8 to b_9. The token placed on location b_7 when transition a_5 fires denotes that the processor is idle. To compute $d(b_8)$, we use the formula

$$d(b_8) = d_{K_1}(b_8) + d_{K_2}(b_8) + \cdots + d_{K_n}(b_8)$$

where the n tokens $K_1, K_2, \ldots, K_n$ reside on location b_8 during the net operation (i.e. n "jobs" are processed by the system).

Definition 12 (Ref 15:56): "Given (E, M_0) and a path, $\gamma_K = b_1, b_2, \ldots, b_n$. The traverse time of a token K through path γ, denoted $D(\gamma_K)$, is the time that K leaves location b_n minus the time K entered location b_1."

The traverse time of a job that does not require a tape, in Figure 13, is the time a job (token) leaves b_{11} minus the time it entered b_1. Still referring to Figure 13, it is noted that the minimum traverse time for the net (the time for a token to reach b_{13} after it has been placed on b_1) is the sum of the transition times $t(a_1), t(a_3), t(a_4), t(a_5), t(a_6)$, and $t(a_8)$ for a job which does not request a tape drive. The actual traverse time for a token, K, representing a "non-tape" job is the sum of the dwell times $d_K(b_1) + d_K(b_4) + d_K(b_6) + d_K(b_8) + d_K(b_9) + d_K(b_{11})$. For any token, K, representing a non-tape job in Figure 12, the following relationship is always true (Ref 15:56):

$$\Sigma\, t(a_i) \leq \Sigma\, d_K(b_j) \qquad \text{where} \quad i = 1,3,4,5,6,8 \quad \text{and} \quad j = 1,4,6,8,9,11$$

Formal Evaluation Net Definition. In the preceding sections, the basic structure of an E-net has been introduced and the concept of net operation has been discussed. In this

section, the token structure will be further defined, which leads to a more complete description of locations. Two mechanisms, transition procedures and resolution procedures, are introduced, which refine the concept of net operation. Also, a special token called an environment variable is introduced which is used to represent a portion of the environment. Attribute tokens, transition procedures, resolution procedures, and environment variables complete the formulation of E-nets. A formal E-net definition is presented, with an example, at the end of this section.

Definition 13 (Ref 15:61): "A simple token is a primitive marker which indicates that a location is full whenever it resides on that location. An attribute token is a unique simple token that has a finite, non-zero number of attributes associated with it."

Attribute tokens require a different notation than simple tokens to indicate a net marking. For simple tokens on a location, b, the marking of b is denoted M(b) = 1 or M(b) = 0 for b full or empty, respectively (see Definition 9). If the token, K, on b is an attribute token, then additional notation is required to indicate the marking of b. If K has n attributes, then the token is denoted K[n] (Ref 15:61). "The i^{th} attribute of K[n] is denoted by K(i), for $1 \leq i \leq n$ (Ref 15:62). (Note that K[n] refers to a token name and K(n) refers to a token attribute value.) Now the marking of location b with token K[n] residing on it is denoted by M(b) = K[n] , and M(b) = 0 if b is empty.

"Attribute tokens impose a data structure on the locations of an evaluation net" (Ref 15:62). A particular location will always receive (provide) a token with a fixed number of attributes. It is not required, however, that all of the locations attached to a particular transition have the same number of attributes. The location specifications in a net description must be redefined to incorporate the concept of location data structure.

Definition 14 (Ref 15:63): Given $E = (L,P,R,A)$, and $b_1 \in L$, then if b_1 may contain only simple tokens, b_1 is declared as "b_1". Otherwise, if b_1 may contain attribute tokens with fixed n attributes, b_1 is declared as "$b_1[n]$".

The i^{th} attribute of a token K[n] residing on b[n] is referred to as M(b(i)). Suppose that a transition, a_i, places token K[n] on location b[m] and $n \neq m$. If g is the smaller of m and n, then the marking of b[m] after a_i fires is (Ref 15:63):

$$\begin{aligned} M(b(1)) &:= K(1) \\ M(b(2)) &:= K(2) \\ &\vdots \\ M(b(g)) &:= K(g) \end{aligned}$$

If $n > m$, then the attributes $m + 1,\ldots, n$ of K[n] are lost. If $n < m$, then the values $M(b(n + 1)),\ldots, M(b(m))$ are undefined (Ref 15:64).

Suppose now that transition a_i is a J-type transition (refer to Definition 4) with input locations $b_1[n]$ and $b_2[n]$ and output location $b_3[n]$. Suppose also that $b_1[n]$ and $b_2[n]$ contain two distinct tokens $K_1[n]$ and $K_2[n]$. After the transition fires, locations $b_1[n]$ and $b_2[n]$ become empty and a token is placed on location $b_3[n]$. What is the identity of the token on $b_3[n]$? This is one example (Ref 15:64) of why the introduction of attribute tokens and the imposition of data structure on locations requires an expanded definition of a transition declaration in order that the function of a transition may be fully interpreted. A procedure is added to the transition declaration to interpret the action of the transition with respect to the attribute tokens place on the transition output location(s).

Definition 15 (Ref 15:65): "A transition procedure describes the action of the environment and the associated locations of the transition on the token. The form will be given in BNF" (Ref 15:65-66) (See Appendix A for an explanation of the BNF form used here):

<transition procedure> ::= [<conditional list>]
<conditional list> ::= <predicate> → (<expression list>:
<conditional list> |
<predicate> → (<expression list>
<Predicate> ::= <Boolean factor> |
<predicate> <Boolean factor>

<Boolean factor> ::= <Boolean secondary> |
<Boolean factor> <Boolean secondary>

<Boolean secondary> ::= <Boolean primary> |
~ <Boolean primary>

<Boolean primary> ::= T | F <relation>

<relation> ::= <right part><relational operator>
<right part>

<relational operator> ::= > | ≥ | = | ≤ | <

<expression list> ::= <expression>;<expression list> |
<expression>)

<expression> ::= <left part><right part>

<right part> ::= <term> | <add op><term> |
<right part><add op><right part>

<term> ::= <factor> | <term><mult op><factor>

<factor> ::= <primary> | <factor> ↑ <primary>

<primary> ::= <unsigned number> |
<variable>(<right part>) |
<variable>

<mult op> ::= * | /

<add op>::= + | -

<left part> ::= <simple variable> :=

<variable> ::= <simple variable> | <number>

<simple variable> ::= M(<location name>(<attribute #>)) |
<environment variable>

<location name> ::= b ∈ {b | b[n] ∈ L or b ∈ L,
b ::= <identifier>}

<attribute #> ::= i ε {i | i is an integer, $1 \leq i < n$}

<environment variable> ::= <identifier>(<attribute #>)

<number> ::= <unsigned number> | + <unsigned number> |
- <unsigned number>

<unsigned number> ::= <decimal number> |
<exponent part> |
<decimal number><exponent part>

<decimal number> ::= <unsigned integer> |
<decimal fraction> |
<unsigned integer><decimal fraction>

<exponent part> ::= $_{10}$ <integer>

<decimal fraction> ::= . <unsigned integer>

<integer> ::= <unsigned integer> | + <unsigned integer>
- <unsigned integer>

<unsigned integer> ::= <digit> | <unsigned integer><digit>

<identifier> ::= <letter> | <identifier><letter> |
<identifier><digit>

<letter> ::= A | B | C | ... | z

<digit> ::= 0 | 1 | 2 | ... | 9

A transition procedure has the form

$$[p_1 \rightarrow (e_{11};\ldots;e_{1n}):\ldots:p_K \rightarrow (e_{K1};\ldots;e_{Km})]$$

where the p_i, $(1 \leq i \leq K)$, are <predicate>'s and the e_{ij} are <expression>'s (Ref 15:66). The p_i are evaluated in numerical order starting with p_1. When the first p_i is

evaluated to true (T), its corresponding expression list, $e_{i1},\ldots,e_{in}$, is executed and then transition procedure evaluation is terminated. If none of the p_i are true, then none of the expression lists are executed. The previous notation describing an E-net structure, Definition 8, must be amended to incorporate the following definition of the entry A in the tuple (L,P,R,A).

Definition 16 (Ref 15:67): The set of transitions, A, is denoted by:

$$A = \{(s,t(a),q) \mid s \in S, t(a) \in T, q \in Q\}$$

S = A finite, non-empty set of transition schema satisfying Definition 4.

T = {t(a) | t(a) is a transition time for transition a *(as specified in Definition 7)*}.

Q = A finite, non-empty set of transition procedures satisfying Definition 15.

The addition of transition procedures requires that part 3) of Definition 5 must now be replaced by the following:

3) "Termination Phase: The transition completes processing, changing the status of the associated locations to agree with the right-hand side of the transition by executing the given transition procedure " (Ref 15:67)

A portion of the environment of a net may be represented by a specialized attribute token known as an environment variable. More precisely, environment variables form part of the interface between a net and its environment.

Definition 17 (Ref 15:68): "An environment variable is an attribute token, K[n], where $n > 0$, that represents the status of a portion of the environment." The set of net environment variables is denoted by:

$$\xi = K_1[n_1], K_2[n_2], \ldots, K_m[n_m] .$$

There are two ways to reference an environment variable. (Refer to Definition 15) An environment variable may appear as a <left part> of a transition procedure <expression>. The result of this type of reference is the alteration of some attribute of the environment variable (Ref 15:68). An environment variable may appear in a <predicate> or <right part> of an <expression>. The result of this type of reference uses the value(s) of the environment variable's attribute(s), but does not alter them (Ref 15:68).

A restriction on net interpretation must be introduced to prevent race conditions with repect to environment variables. No two transitions may terminate simultaneously or change from pseudo enabled to enabled simultaneously if a common environment variable is involved. A transition may not change from pseudo enabled to enabled at the same instant that another transition terminates if a common environment variable is involved (Ref 15:96).

Another portion of the net-environment interface has already been introduced: peripheral resolution locations (Definition 3). Referring to Definition 4, when an X or Y transition fires and the transition's input r-location is also a peripheral location, the firing of the transition leaves the status of the resolution location undefined. Before the transition can become enabled again the environment has to define the resolution location's status (Definition 6). The mechanism for defining the status of peripheral resolution locations is the resolution procedure.

Definition 18 (Ref 15:69): "A resolution procedure is an expression

$$\Psi(r) = r :[\langle predicate\rangle \rightarrow M(r) := s : \langle predicate\rangle \quad M(r) := 1 - s]$$

where $s \in \{0,1\}$ and r is the label of the peripheral resolution location. Denote the set of resolution procedures as

$$\Psi = \{\Psi(r) \mid \Psi(r) \text{ is a resolution procedure and } r \in P \quad R\}$$

If the first <predicate> is true, then M(r) is set to s. Else, if the second <predicate> is true, then M(r) is set to 1 - s. Otherwise, the setting of r remains undefined" (Ref 15:69).

A resolution procedure is only evaluated when the transition is pseudo enabled (see Definition 6). If the resolution location status is still undefined after the resolution procedure is evaluated, then the value of at least one argument of one of the <predicate>'s must change before the resolution procedure is evaluated again (Ref 15:69).

All of the E-net constructs and mechanisms have been introduced, and it is now possible to formulate a final net definition which provides the E-net structure, the initial marking, and the environment interface. Following the general definition, the formal net description of the net in Figure 13 will be presented.

Definition 19 (Ref 15:70): "An evaluation net is a tuple,

$(E, M_0(\xi, \Psi))$ such that

$E = (L,P,R,A)$ is an evaluation net structure satisfying Definition 8.

M_0 is an initial marking (Definition 9).

ξ is a set of environment variables (Definition 17).

Ψ is a set of resolution procedures (Definition 18).

Before presenting a formal net description of the net in Figure 13, it is desireable to refine the net itself by incorporating attribute tokens in the net model to represent the "jobs" in the "system". A job token will have three attributes:

1) K(1) : Central processor utilization time.
2) K(2) : Tape I/O time.
3) K(3) : 1 if the tape drive is required, 0 otherwise.

Assuming that processor utilization and tape I/O are fully overlapped, the processor is allocated to a job for max (K(1), K(2)) time units. Other assumptions, related to tape drive allocation,

are that it takes 45 seconds to fetch a tape as represented by $t_1(a_1)$ and it takes 15 seconds to mount or dismount a tape as represented by $t(a_2)$ and $t(a_7)$, respectively. The formal net definition of the refined model of Figure 13 is as follows (Ref 15:71-73):

$$E = (L,P,R,A),$$

$$R = \{r_1,r_2,r_3,r_4\}$$

$$P = \{b_1[3],b_{13}[3]\} \cup R$$

$$L = \{b_2,b_3[3],b_4[3],b_5[3],b_6[3],b_7,b_8[3],$$
$$b_9[3],b_{10}[3],b_{11}[3],b_{12}[3]\} \cup P$$

$$A = \{a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8\}$$

$$a_1 = (X(r_1,b_1[3],b_4[3],b_3[3]),\ (0,45 \text{ seconds}),$$
$$[(M(r_1) = 1) \rightarrow (M(b_3[3]) := M(b_1[3])):$$
$$T \rightarrow (M(b_4[3]) := M(b_1[3]))])$$

$$a_2 = (J(b_2,b_3[3],b_5[3]),\ 15 \text{ seconds},$$
$$[T \rightarrow (M(b_5[3]) := M(b_3[3]))])$$

$$a_3 = (Y(r_2,b_4[3],b_5[3],b_6[3]),\ (0,0),$$
$$[(M(b_4[3]) \neq 0 \wedge M(b_5[3]) = 0) \rightarrow$$
$$(M(b_6[3]) := M(b_4[3])):$$
$$(M(b_4[3]) = 0 \wedge M(b_5[3]) \neq 0) \rightarrow$$
$$(M(b_6[3]) := M(b_5[3])):$$
$$M(r_2) = 0 \rightarrow (M(b_6[3]) := M(b_4[3])):$$
$$T \rightarrow (M(b_6[3] := M(b_5[3]))])$$

$a_4 = (J(b_6[3], b_7, b_8[3]), 0,$

$[T \rightarrow (M(b_8[3]) := M(b_6[3]))])$

$a_5 = (F(b_8[3], b_9[3], b_7), \underline{max}(M(b_8(1)), M(b_8(2))),$

$[T \rightarrow (CPU(1) := CPU(1) + M(b_8(1));$

$TIO(1) := TIO(1) + M(b_8(2));$

$M(b_7) := 1)])$

$a_6 = (X(r_3, b_9[3], b_{11}[3], b_{10}[3]), (0,0),$

$[M(r_3) = 0 \rightarrow (M(b_{11}[3]) := M(b_9[3])):$

$T \rightarrow (M(b_{10}[3]) := M(b_9[3]))])$

$a_7 = (F(b_{10}[3], b_2, b_{12}[3]), 15 \text{ seconds},$

$[T \rightarrow (M(b_2) := 1;$

$M(b_{12}[3]) := M(b_{10}[3]))])$

$a_8 = (Y(r_4, b_{11}[3], b_{12}[3], b_{13}[3]), (0,0),$

$[(M(b_{11}[3]) \neq 0 \wedge M(b_{12}[3]) = 0) \rightarrow$

$(M(b_{13}[3]) := M(b_{11}[3])):$

$(M(b_{11}[3]) = 0 \wedge M(b_{12}[3]) \neq 0) \rightarrow$

$(M(b_{13}[3]) := M(b_{12}[3])):$

$M(r_4) = 0 \rightarrow (M(b_{13}[3]) := M(b_{11}[3])):$

$T \rightarrow (M(b_{13}[3]) := M(b_{12}[3]))])$

$M_0(b_2) = M_0(b_7) = 1;\ M_0(b_i[3]) = 0,$

(for all $1 \leq i \leq 13$,
$i \neq 2,\ i \neq 7$).

ξ = {CPU[1], TIO[1]}

Ψ = {$\Psi(r_1)$, $\Psi(r_2)$, $\Psi(r_3)$, $\Psi(r_4)$}

$\Psi(r_1) = r_1$:[M(b_1(3)) = 0 → M(r_1):= 0:

T → M(r_1):= 1]

$\Psi(r_2) = r_2$:[T → M(r_2):=1]

$\Psi(r_3) = r_3$:[M(b_9(3)) = 0 → M(r_3);= 0:

T → M(r_3):= 1]

$\Psi(r_4) = r_4$:[T → M(r_4): = 1]

This new definition of the net in Figure 13 provides a full interpretation of the graph. When a token is placed on location b_1, transition a_1 is pseudo enabled. Resolution procedure $\Psi(r_1)$ is then evaluated. Based on the value of attribute three of the token, M(r_1) will be set to 0 or 1. Assigning the value 0 or 1 to r_1 enables transition a_1. Transition a_1 fires, with firing time equal to 45 or 0 seconds, moving the token on b_1 to b_4 or b_3, respectively. The environment variables, CPU[1] and TIO[1], are updated each time transition a_5 fires, providing the system performance measures of total CPU and tape drive usage. Attribute tokens, transition procedures, and resolution procedures greatly enhance the performance measurement capabilities of E-nets while providing a clear and definitive method of executing the net.

Macro Structures. It is sometimes desireable to be able

to "compress" a redundant sequence of transitions and locations or to replace a cluster of nodes with a simple structure which is a clear and more concise representation. Macro E-nets are structures which provide just such a capability (Ref 16,17). Macro structures can be derived to meet the needs of the net under consideration. However a "standard" macro structure, a priority-in queue, will be presented since they appear in some of the nets in Chapter III. Formal derivations of several types of macro structures can be found in Reference 15 (113-126) and 17.

Figure 14 is a priority-in queue of length four composed of evaluation net primitives. Each non-resolution location may accept an attribute token $K_i[1]$ with one attribute , which represents the token's priority. Location q_4 is the tail of the queue and q_1 is the head of the queue. Assume the following net marking:

$$M(q_1) = K_1[1] \quad , \quad K_1(1) = 3 \quad ;$$

$$M(q_2) = K_2[1] \quad , \quad K_2(1) = 1 \quad ;$$

$$M(q_3) = M(q_4) = M(b_1) = M(b_2) = M(u_1)$$

$$= M(d_1) = M(d_2) = M(d_3) = 0 \quad ;$$

$$M(r_i) = M(s_i) = \phi \quad , \text{ for } i = 1,2,3 \quad .$$

Now place $K_3[1]$ on b_1, with $K_3(1) = 2$. It is required to place $K_3[1]$ on q_2 and move $K_2[1]$ back to q_3. The following markings and transition firings will perform the needed actions (Ref 15:115-116):

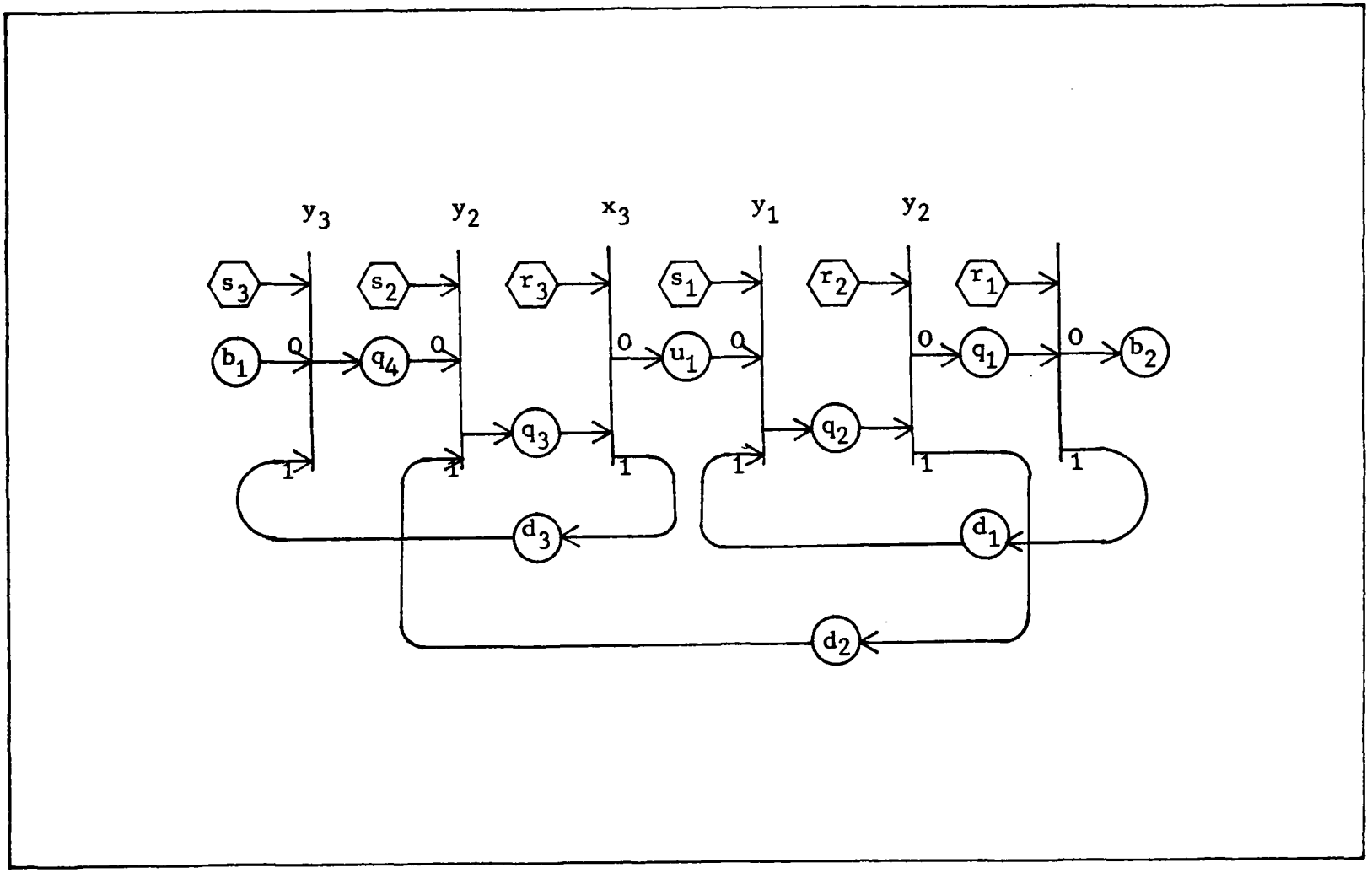


Fig. 14 Priority-In Queue of Length Four (Ref 15:112)

Set $M(s_3) = 0$; Fire y_3, (move $K_3[1]$ forward).

Set $M(s_2) = 0$; Fire y_2, (move $K_3[1]$ forward).

Set $M(r_3) = 0$; Fire x_3, (move $K_3[1]$ forward).

Set $M(r_2) = 1$; Fire x_2, (move $K_2[1]$ backward).

Set $M(s_2) = 1$; Fire y_2, (move $K_2[1]$ to q_3).

Set $M(s_1) = 0$; Fire y_1, (move $K_3[1]$ to q_2).

Now, $K_1[1]$ resides on q_1, $K_3[1]$ resides on q_2, and $K_2[1]$ resides on q_3, and q_4 is empty. Since setting the resolution locations to the appropriate status causes the associated transitions to fire it is sufficient to only list the resolution location settings that will remove the token from the head of the queue (placing it on location b_2) (Ref 15:117):

Set $M(r_1) = 0$.

Set $M(r_2) = 0$.

Set $M(r_3) = 0$.

Set $M(s_1) = 0$.

Now, $K_3[1]$ resides on q_1, $k_2[1]$ resides on q_2, and q_3 and q_4 are empty.

The formal definition of a priority-in queue must still be stated in terms of the primitive constructs when the overall net definition is declared. However, the substitution of the structure in Figure 15 for the net in Figure 14 would

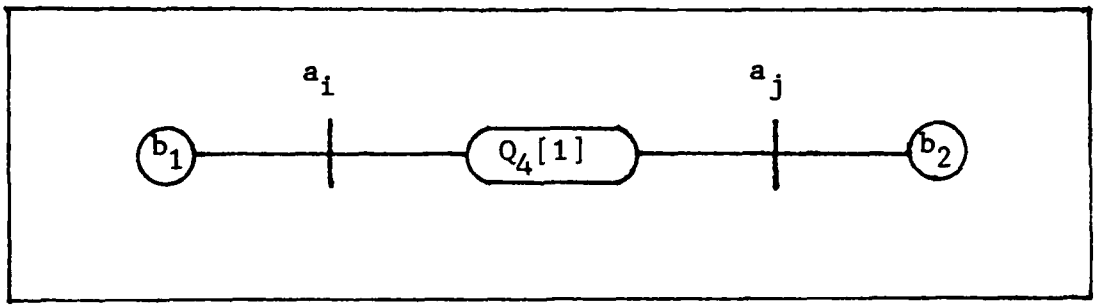


Fig. 15 Priority-In Queue (Ref 17:20)

present a much clearer picture of a net incorporating a priority-in queue of length four and tokens with one attribute.

Summary. E-nets are characterized as marked, interpreted, directed graphs. An E-net is composed of two sets of nodes, transitions and locations, which are interconnected by directed arcs. Tokens are markers which can reside on the locations (a particular location may hold at most one token). A token may have a finite set of attributes with values assigned to the attributes. Net operation (or execution) is reflected in the movement of tokens among the location nodes in the net. Tokens are moved by the "firings" of transition nodes. When the status of the locations attached to a transition satisfy defined conditions, the transition is enabled and subsequently fires, removing tokens from a subset of its input locations and placing tokens on a subset of its output locations. A transition is specified by a tuple, (s,t(a),q), where s is the transition type, t(a) is the transition "firing" time duration, and q is a transition procedure which interprets the action of the transition on the tokens involved in a particular

firing. A resolution location is a special location which provides a conflict avoidance mechanism for a transition which selects a token to be removed from (placed on) either of two input (output) locations when the transition fires. The net-environment interface is specified by peripheral non-resolution locations, resolution procedures, and environment variables. A peripheral location has only one transition connected to it by one arc. Peripheral non-resolution locations provide an input (output) for tokens from (to) the environment to (from) the net. A resolution procedure is a mechanism for defining the status of peripheral resolution location. An environment variable is a special attribute token which represents the status of a portion of the environment. Environment variables can be referenced by resolution procedures and referenced and changed by a transition procedure. In the next chapter, an E-net graph model of the DAIS will be developed and a formal description of the graph presented.

III Model Construction

Prefatory Activities

Prior to the actual construction of a system model, guidelines should be established which will serve to keep the modeling effort "on track". These guidelines can often be separated into functionally diverse groups which may be developed by distinct activities. Three activities preceded the actual construction of an E-net model of the DAIS: a) determination of which DAIS components and/or functions to model, b) determination of the lowest level of detail to be modeled, and c) establishment of a set of model requirements. There is some overlap and interdependency among these three activities; the level of detail may be restricted or bounded by which system functions are modeled, and some model requirements will be established or tailored to consider what system functions are modeled.

What to Model. In the introduction of this report it was stated that a model is desired which could be used for high level analyses of proposed distributed processor avionics computer systems. For this investigation, a two processor configuration of the DAIS, master processor plus one remote processor (Ref 1,24), is modeled. The distributed architecture, system control, and concurrent processing activity of the DAIS are of primary interest. The processors, their

associated BCIUs, and the data bus should be modeled. The Remote Terminals and other avionics components (communications devices, weapons, altimeters, etc.) are only input and/or output devices to the computer system. Therefore, they do not need to be physically represented in the model; only the transmission (reception) of messages to (from) these other components from (by) the processors needs to be modeled.

System control refers to one of the major functions of the DAIS Executive. Data bus control is another major function of the Executive. Both of these functions need to be modeled in order to analyze system operations by using the model. Another type of control function that is not so obvious is the control of a processor's activity. Which task is selected for execution on a processor depends on the task's position in a prioritized list of tasks that are ready for execution. Some tasks (Normal Mode Tasks) can have their execution suspended if a higher priority task becomes ready to execute. Other tasks (Privileged Mode Tasks and Executive actions) run to completion once their execution is begun. Overriding the task execution function is the hardware interrupt capability of the BCIU to its associated processor. Also, the activity (i.e. utilization) of the processors and their BCIUs should be reflected in the model, to support efforts to analyze the impact of adding or deleting processors from a distributed processor configuration. Finally, the interaction of a processor with its BCIU and the differences in operation between Master and Remote BCIUs should be revealed in the

model, as points of major interest in a system analysis.

In summary, then, the components, functions, and activities of the DAIS to be modeled are:

1) Distributed architecture (processors, BCIUs, and Data Bus),
2) System control,
3) Data Bus control,
4) Task control,
5) BCIU interrupts to processors,
6) Task priority scheme,
7) Processing activity,
8) BCIU and Data Bus activity, and
9) Processor - BCIU interaction.

Level of Detail. The level of detail at which to model a system depends, of course, on the intended use of the model. The inherent capabilities of the modeling schema used may limit the degree of detail which can be represented by a particular model type. For this investigation, a level of detail is desired which illustrates the DAIS components, functions, and activities listed in the previous section. This level of detail is determined to be the task level. Tasks and Executive actions are represented by tokens, whose flow through the E-net graph will simulate the flow of tasks in the DAIS.

Model Requirements. A set of model requirements is established which express the desired capabilities of the E-net model representation of the DAIS. These requirements are, admittedly, somewhat subjective, but they represent the intent

of this investigation. The model requirements are:

1) The E-net graph is a pictoral representation of the DAIS structure.

2) The functions and activities listed in the section "What to Model", above, are incorporated into the graph structure or are revealed by the graph interpretation.

3) The E-net model represents the DAIS at a "task flow" level of detail.

4) Interpretation of the E-net graph simulates the operation of the DAIS with respect to task flow and the system control and bus control functions.

5) Analysis of the interpretation of the E-net graph can be used to provide answers to questions about DAIS performance (especially with respect to processor utilization and bus activity).

Model Development

The guidelines provided by the prefatory activities provided a basis on which to begin the model construction. The approach used for model construction was top-down development. The first graph constructed represents the DAIS at the highest level. Succeeding graphs representing successively greater levels of detail were constructed until a level of detail was reached which represented the flow of tasks in the DAIS. Three graphs were constructed in all; they are presented in the following sections in the order of their development. For the first two graphs, only the basic E-net description is given, since these graphs were not to be interpreted. The first and second graphs constructed do not meet requirements 2) and 3) under "Model Requirements", above, but they do represent the DAIS at high levels of detail and they give insight

into the step-by-step method of developing the E-net graph model which is the end product of this investigation.

Top Level Model. The first stage in the top-down model development was the construction of a very high level graph representation of the DAIS. This "top level" model is seen in Figure 16. The nodes in the graph are defined as follows:

a_1 : add job to master processor queue
a_2 : allocate the master processor to a job
a_3 : deallocate the master processor
a_4 : dispose of last job executed on master processor
a_5 : place message on bus
a_6 : transfer message from bus to a processor
a_7 : add job to remote processor queue
a_8 : allocate the remote processor to a job
a_9 : deallocate the remote processor
a_{10}: dispose of last job executed on remote processor
b_1 : a job is ready to execute
b_2 : master processor is allocated
b_3 : master processor is deallocated
b_4 : job execution is completed
b_5 : master processor message awaits transmission
b_6 : last job did not generae a bus message
b_7 : remote processor message awaits transmission
b_8 : the bus is busy
b_9 : a message for the master processor has been received
b_{10}: a message for the remote processor has been received

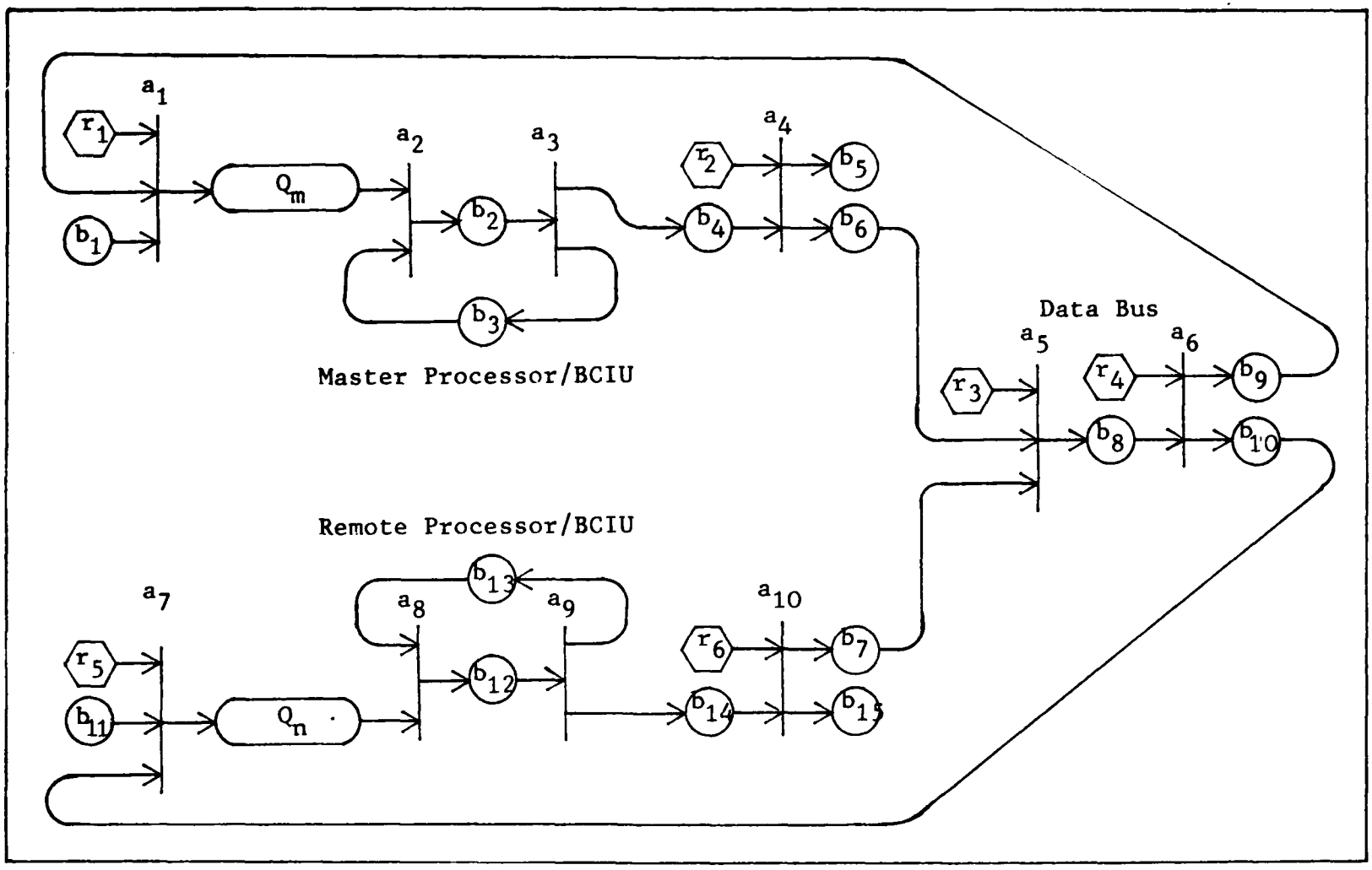


Fig. 16 High Level E-Net Graph of the DAIS

b_{11}: a job is ready to execute

b_{12}: remote processor is allocated

b_{13}: remote processor is deallocated

b_{14}: job execution is completed

b_{15}: last job did not generate a bus message

Q_m : priority-in queue for master processor

Q_n : priority-in queue for remote processor

r_1 : choose jobs for input to master processor queue

r_2 : detect bus messages

r_3 : collect messages for bus transmission

r_4 : choose processor to receive message

r_5 : choose jobs for input to remote processor queue

r_6 : detect bus messages.

In this top level model, the environment places jobs on locations b_1 and b_{11} for the master processor and remote processor, respectively. $Q_m[j]$ and $Q_n[j]$ represent priority-in queues of executable jobs. Jobs work their way to the head of their queues and eventually are executed. Some jobs may generate a bus message. Jobs which do not generate bus messages exit the graph at nodes b_5 and b_{15}. If a bus message is generated, a token is placed on location b_6 or b_7, appropriately. For example, if a job represented by token K is placed on location b_1, and this job needs to send the results of its computation to the remote processor, the path followed by K is

$b_1, Q_m, b_2, b_4, b_6, b_8, b_{10}, b_{12}, Q_n, b_{12}, b_{14}, b_{15}$.

The formal definition of the graph in Figure 16 is as follows:

$$E = (L,P,R,A)$$

$$R = \{r_1,r_2,r_3,r_4,r_5,r_6\}$$

$$P = \{b_1,b_5,b_{11},b_{15}\} \cup R$$

$$L = \{b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_{10},b_{12},b_{13},Q_m,Q_n\} \cup P$$

$$A = \{a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8,a_9,a_{10}\}$$

$$a_1 = (Y(r_1,b_9,b_1,Q_m) , (t_0(a_1), t_1(a_1)))$$

$$a_2 = (J(Q_m,b_3,b_2) , t(a_2))$$

$$a_3 = (F(b_2,b_3,b_4) , t(a_3))$$

$$a_4 = (X(r_2,b_4,b_5,b_6) , (t_0(a_4) , t_1(a_4)))$$

$$a_5 = (Y(r_3,t_5,t_7,t_8) , (t_0(a_5) , t_1(a_5)))$$

$$a_6 = (X(r_4,b_8,b_9,b_{10}) , t_0(a_6) , t_1(a_6)))$$

$$a_7 = (Y(r_5,b_{10},b_{11},Q_n) , (t_0(a_7) , t_1(a_7)))$$

$$a_8 = (J(Q_n,b_{13},b_{12}) , t(a_8))$$

$$a_9 = (F(b_{12},b_{13},b_{14}) , t(a_9))$$

$$a_{10} = (X(r_6,b_{14},b_7,b_{15}) , (t_0(a_{10}) , t_1(a_{10})))$$

The graph in Figure 16 provides a very high level of observation of the DAIS structure. The distributed architecture and parallel processing capability are visible. Some operational features and system functions and components are not visible at this level of detail, such as BCIU operation, BCIU interrupt capability to its processor, and the processor-to-BCIU programmed I/O interface. Jobs are considered to

execute to completion once they are begun.

Intermediate Level Model. The next step in the modeling process was the construction of an E-net graph of the DAIS which represents the operation and structure of the DAIS at a finer level of detail than that provided by the top level model. This next step model, or intermediate level graph, is seen in Figure 17. The nodes in this graph are defined and the formal definition of the net is presented in Appendix B.

The distributed architecture and parallel processing capability are revealed in the intermediate level model of the DAIS. BCIU operation and portions of the processor-BCIU interface are also visible at this level of observation. The bottom half of the figure on sheets 1 and 2 of Figure 17 represent the master and remote BCIUs, respectively. Sheet 3 of Figure 17 is the portion of the E-net graph which represents the data bus and its interconnections to the BCIUs and RT (at this level, nodes b_{33} and b_{40} represent the bus connections of all RTs and directly connected bus compatible avionics subsystems).

The master processor executes jobs (which can represent tasks at this level of detail) it receives at location b_1 and processes asynchronous requests it receives at location b_{15}. A processing activity runs to completion unless a higher priority activity becomes ready to execute (and works its way to the head of the processor queue), in which case the

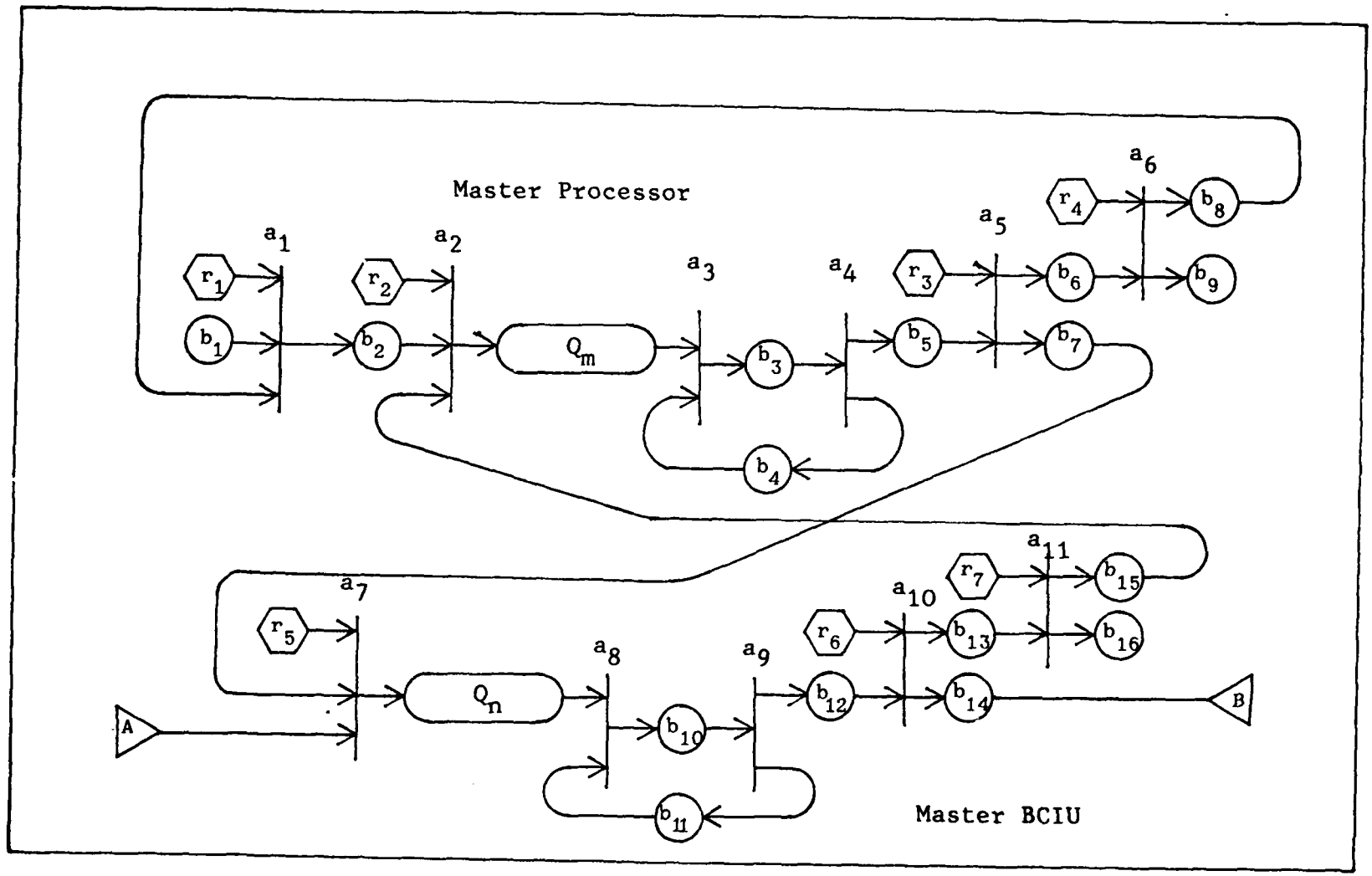


Fig. 17 (1 Of 3) Intermediate Level E-Net Graph of the DAIS

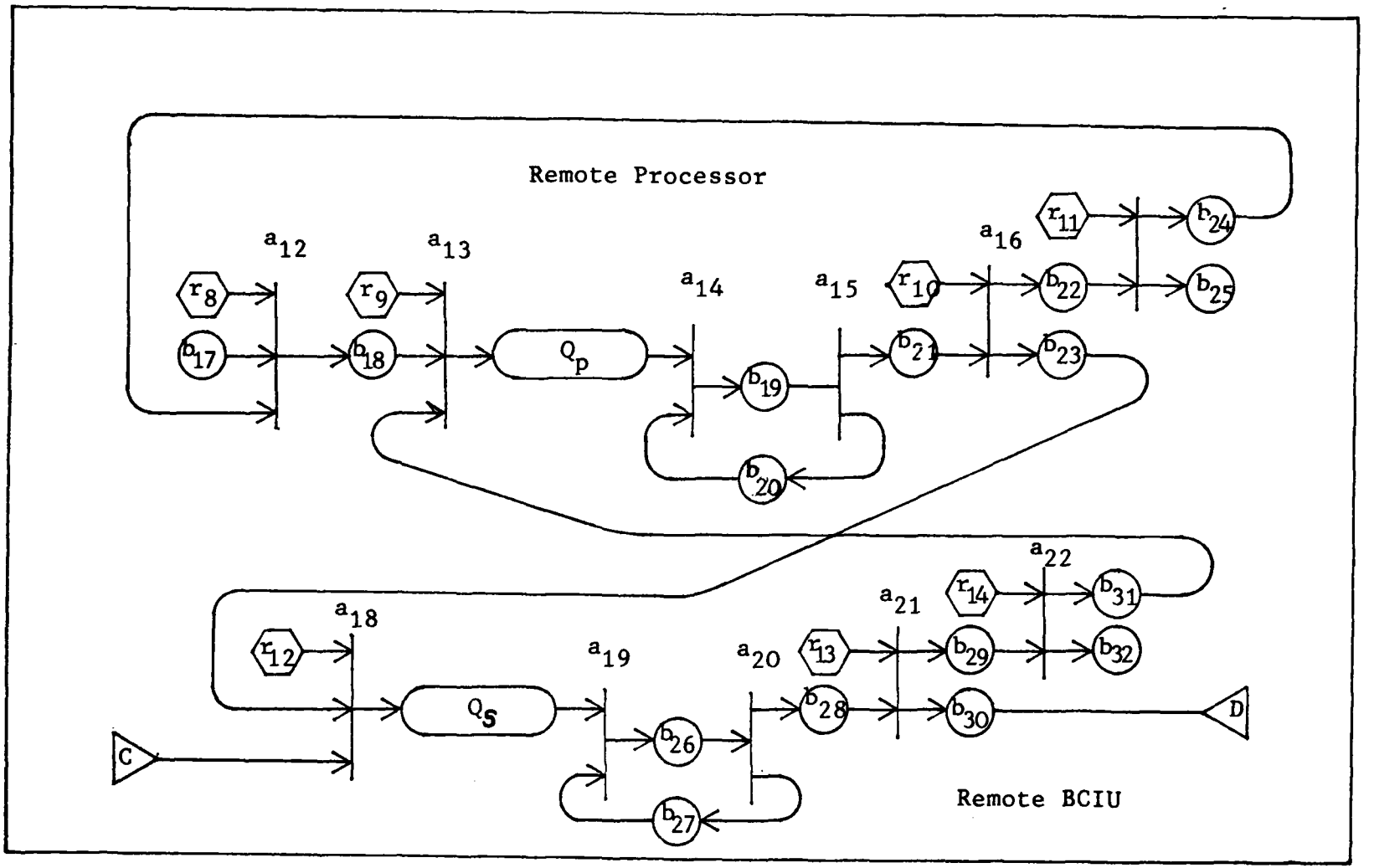


Fig. 17 (2 Of 3) Intermediate Level E-Net Graph of the DAIS

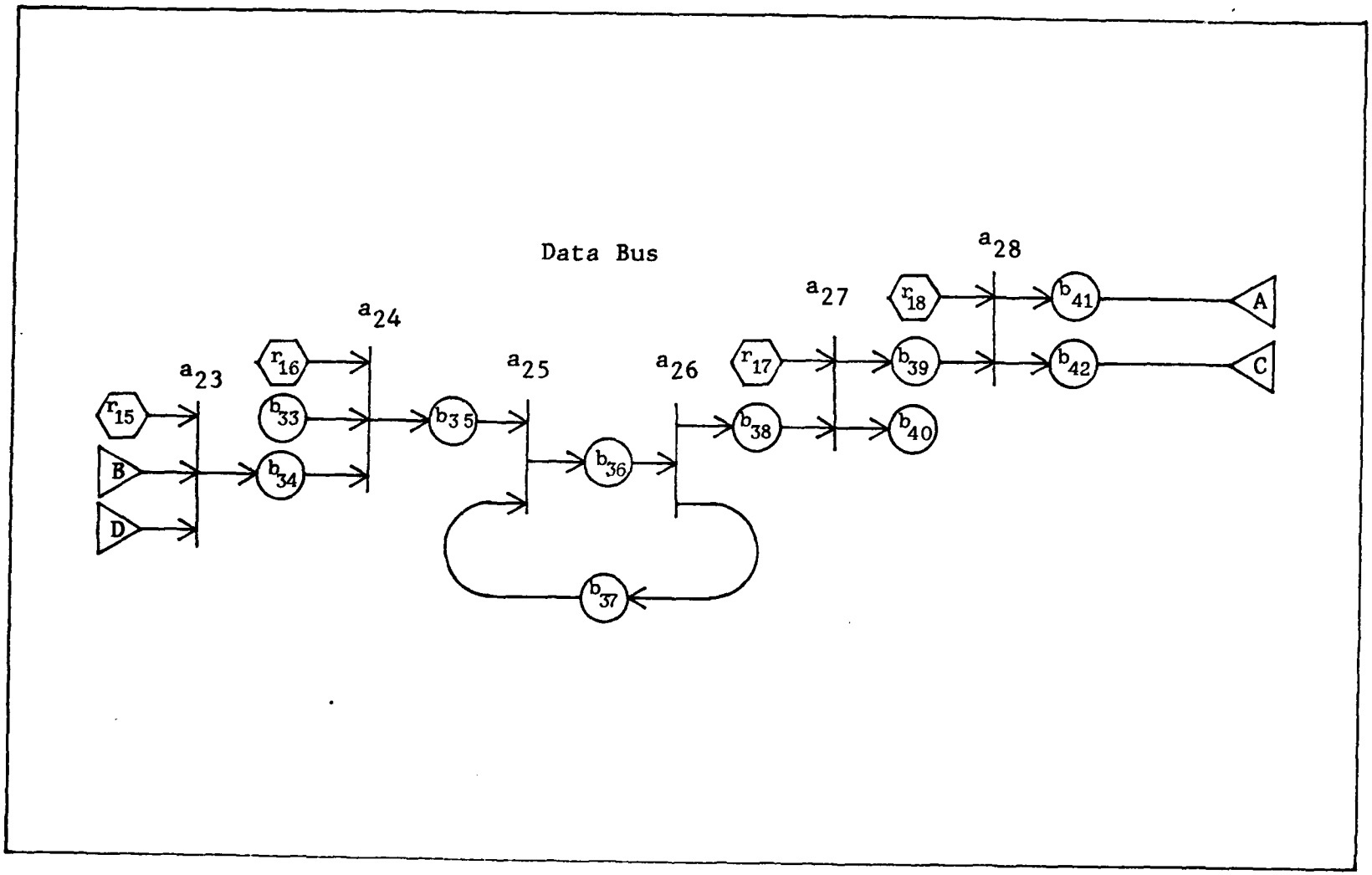


Fig. 17 (3 Of 3) Intermediate Level E-Net Graph of the DAIS

currently executing job is suspended. The suspended job is fed back into the queue (b_8 to b_2 to Q_m) and the processor is allocated to the highest priority job in the queue. If a job generates a bus message, then a token is placed on location b_7 after the job is executed (at this level of observation a job may only generate a message if it completes execution, since the firing of transition a_5 can only place a token on b_6 or b_7). Placing tokens on b_7 and b_{23} represents PIO operations by the master and remote processors, respectively.

Locations b_{14} and b_{41} and locations b_{30} and b_{42} represent portions of the master and remote BCIUs, respectively. Locations b_{41} and b_{42} receive messages from the data bus for their respective BCIUs. The BCU portion of the BCIU processes messages from the bus and its associated processor. The BCU activity is represented by transitions a_8 and a_9 for the master BCIU and transitions a_{19} and a_{20} for the remote BCIU. Messages to be transmitted onto the bus are placed on locations b_{14} and b_{30}.

Although the level of observation provided by the E-net in Figure 17 reveals more operational and functional characteristics than the top level graph does, some functions and control aspects are still not visible. Specifically, the task control, bus control, and system control functions, the BCIU interrupt capability to the processor, and the task priority

scheme are not visible.

Task Level Model. A task flow level of detail is represented by the E-net graph in Figure 18. The formal definition of this net and the definitions of the nodes in the net are presented in Appendix C.

The task level E-net graph in Figure 18 is a pictoral representation of the DAIS. Sheets 1 and 2 of Figure 18 represent the master processer, sheets 4 and 5 represent the remote processor, and sheet 6 represents the remote BCIU. The upper half of sheet 3 represents the master BCIU and the lower half represents the data bus.

One of the assumptions made during development of the task level model (and the previous two models) is that there is no major difference in how tasks are executed in the master and remote processors. Any differences are in the functions of the tasks being executed in each processor. Another assumption is that the Master Interrupt Handler is functionally identical to the Local Interrupt Handler, at the task flow level of detail, with respect to how interrupts from the BCIU are processed. Unavailability of detailed documentation on the Master Executive is the motive for these assumptions. The effect of these assumptions is that the portions of the task level E-net graph representing the master and remote processors are identical.

System control is provided by a collection of functions (see page 20). Error management, configuration management, and mass memory management were not incorporated into the

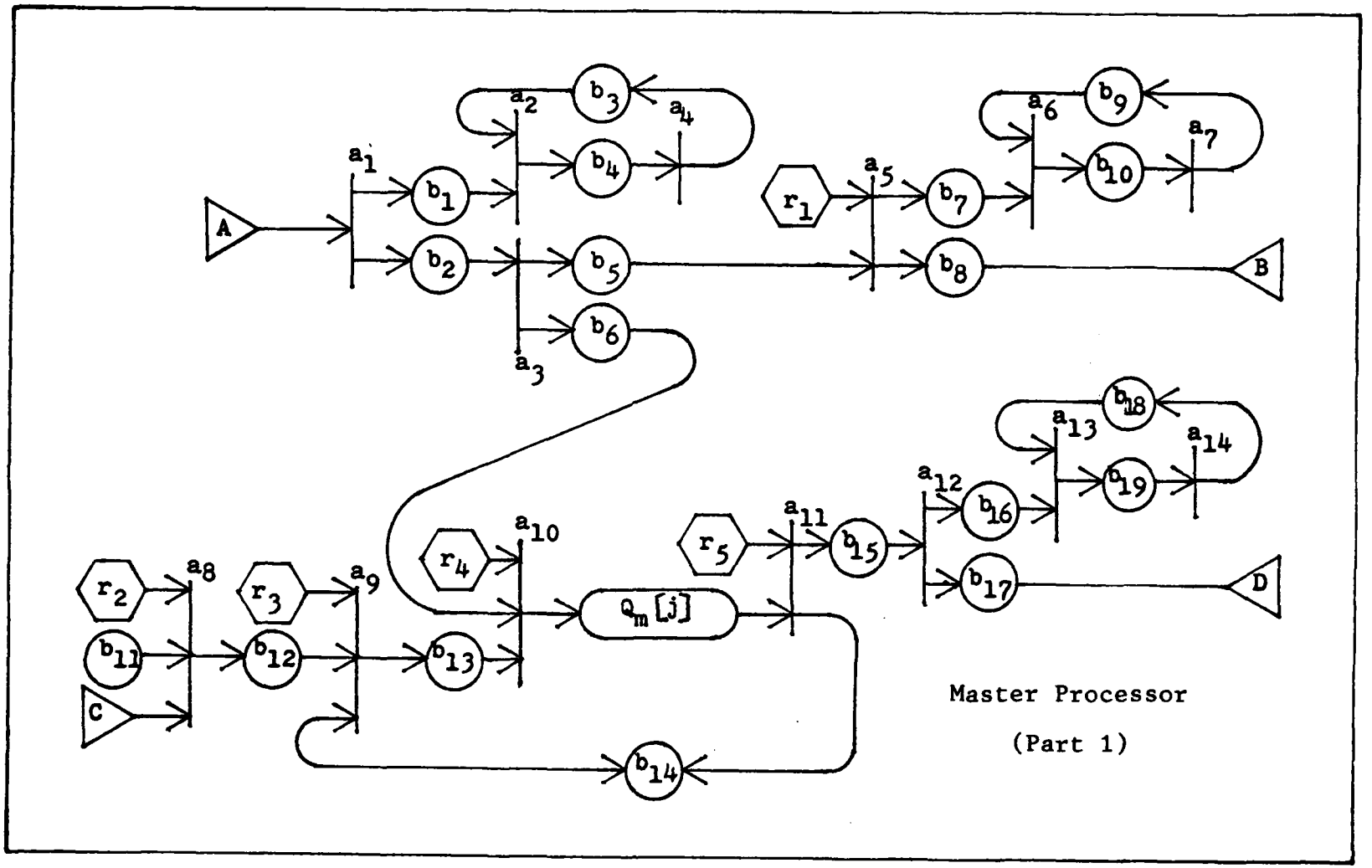


Fig. 18(1 of 6) Task Level E-Net Graph of the DAIS

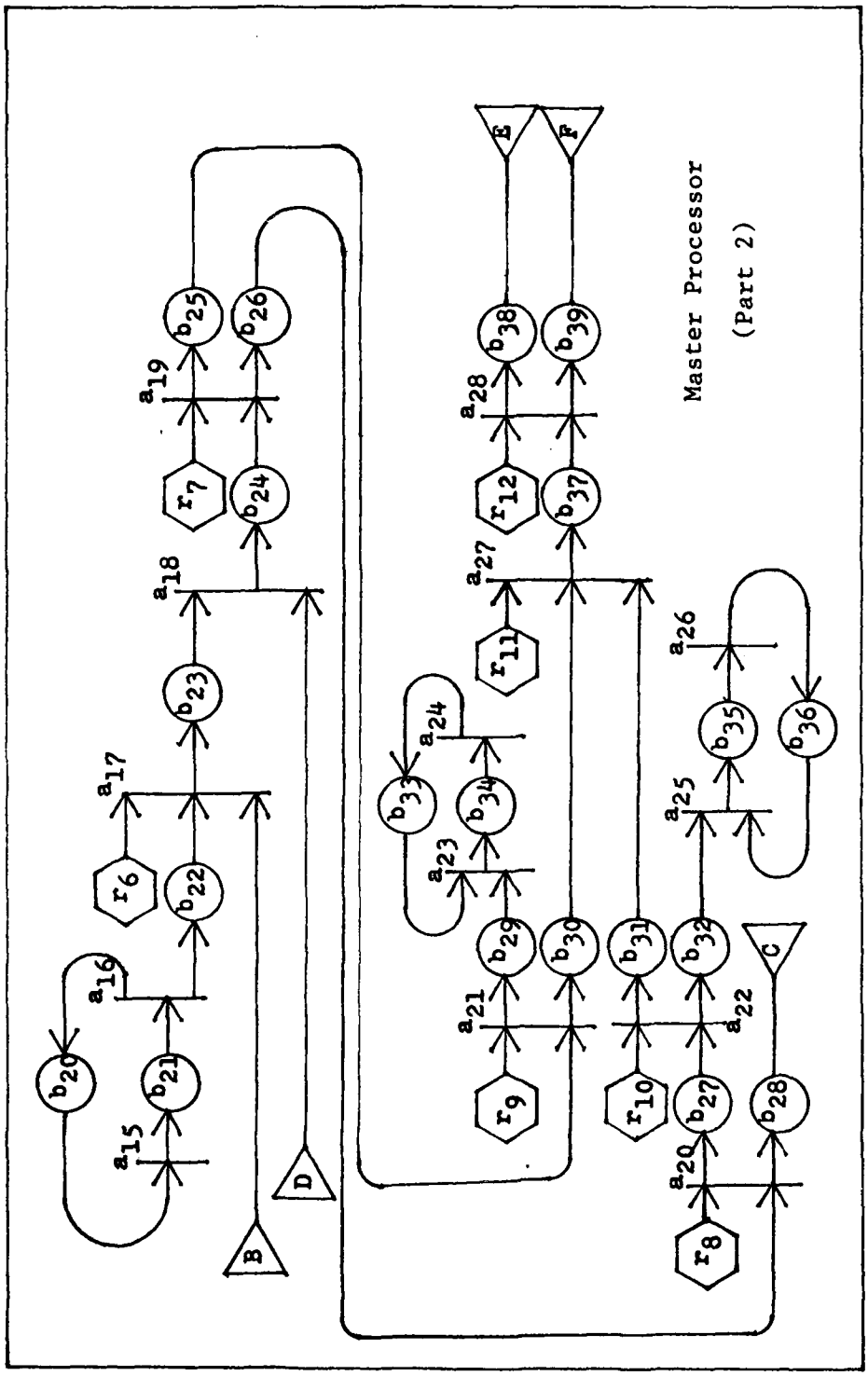


Fig. 18 (2 Of 6) Task level E-Net Graph of the DAIS

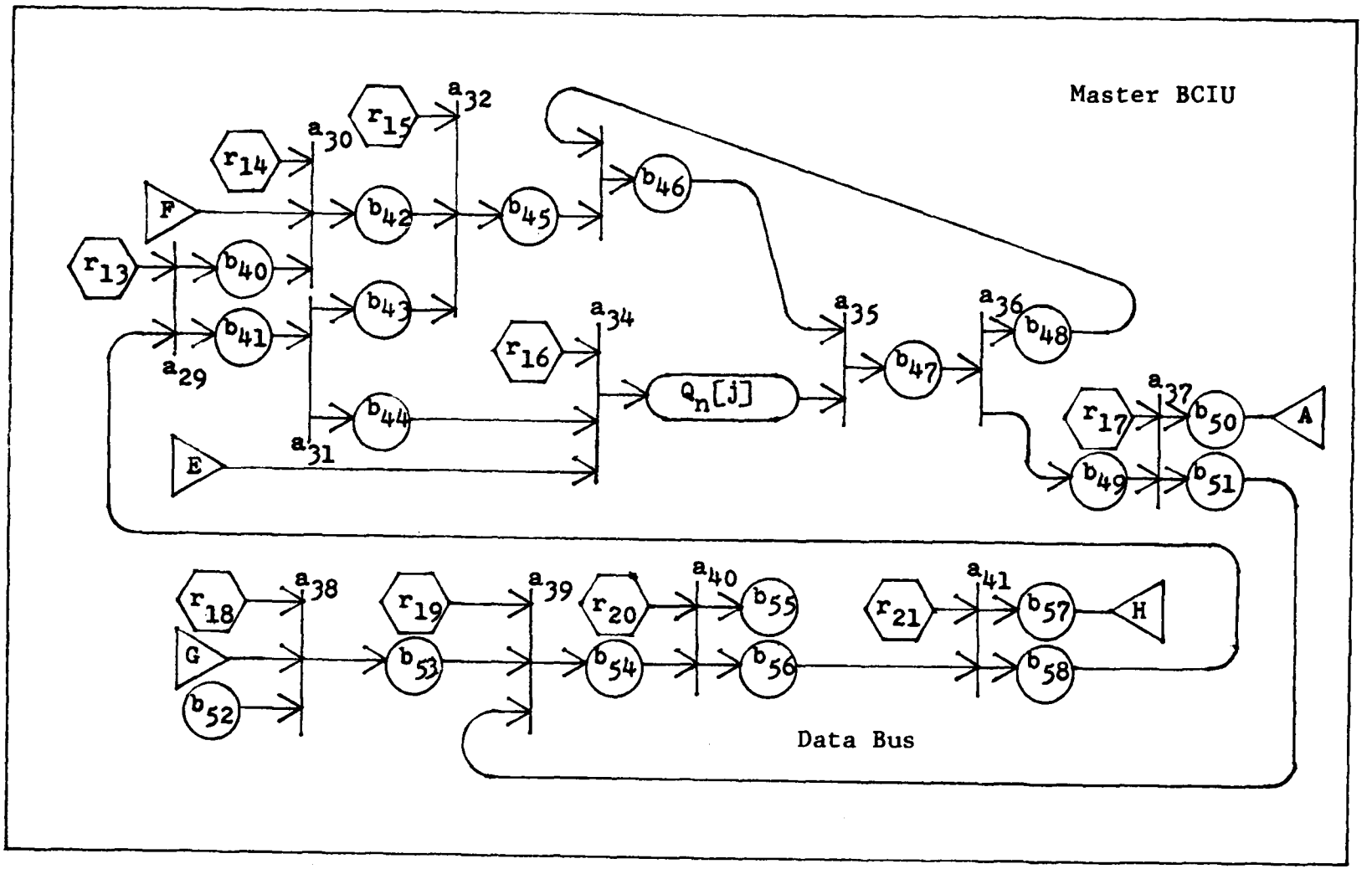


Fig. 18 (3 Of 6) Task Level E-Net Graph of the DAIS

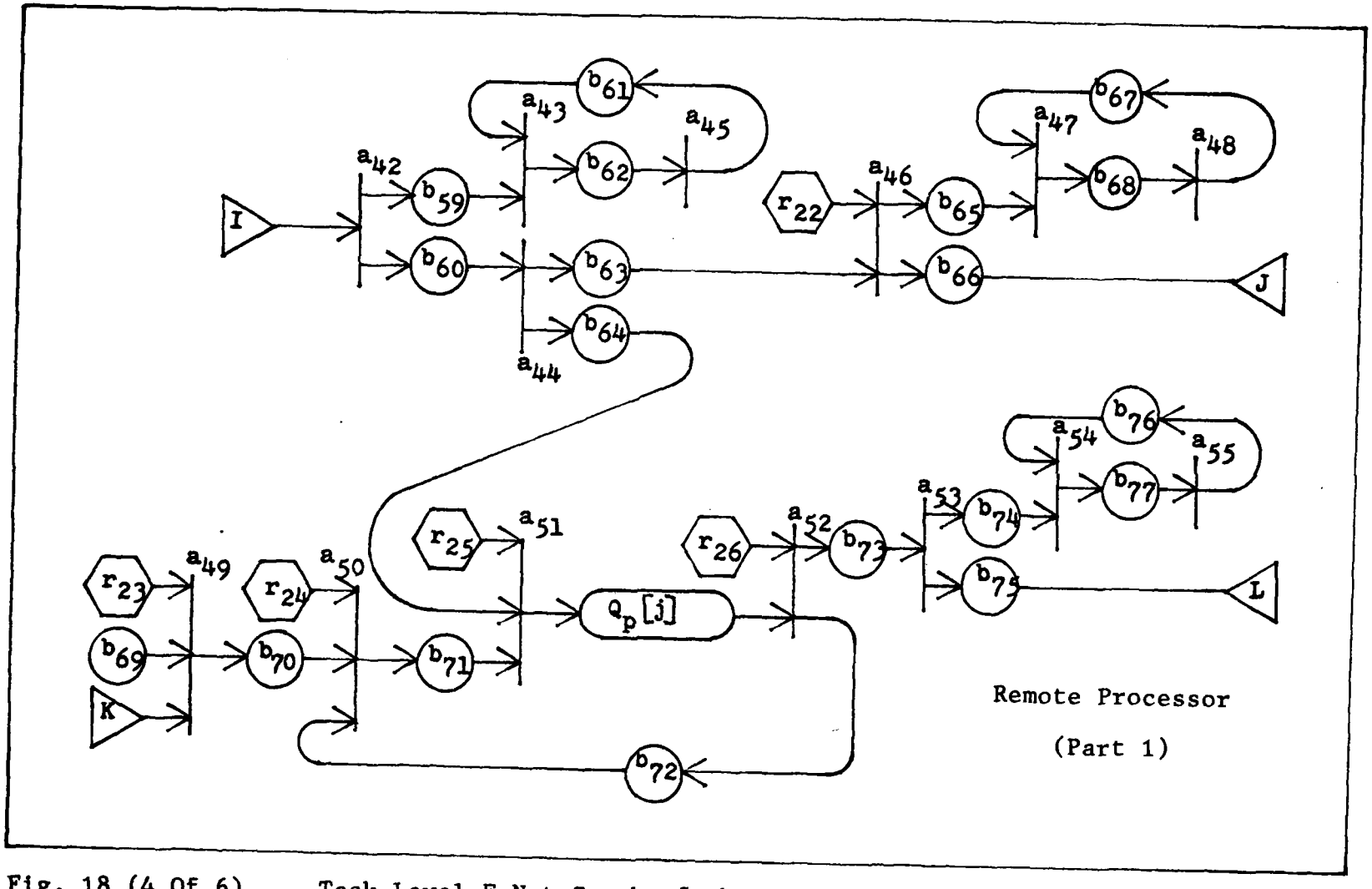


Fig. 18 (4 Of 6) Task Level E-Net Graph of the DAIS

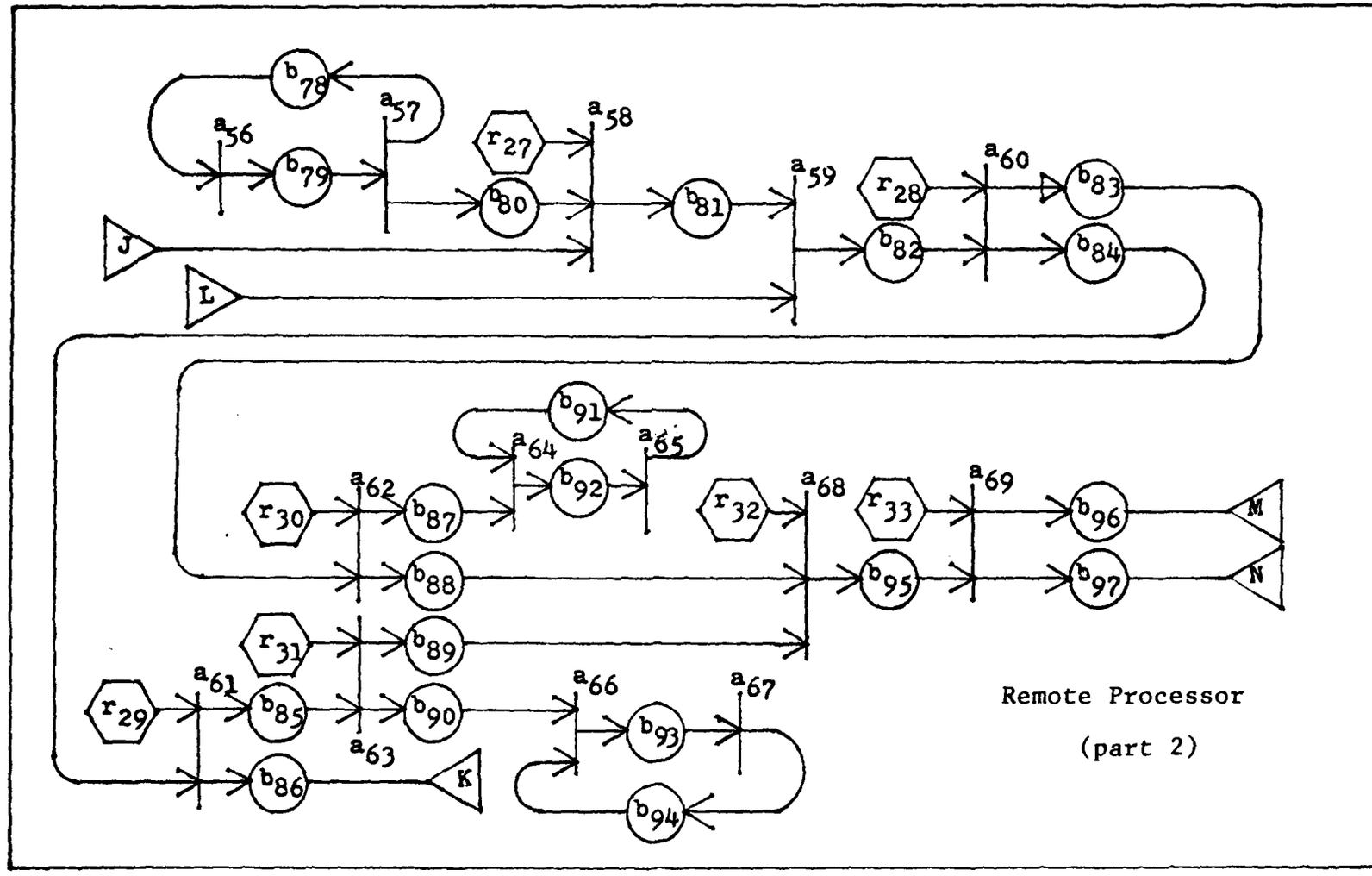


Fig. 18 (5 of 6) Task Level E-Net Graph of the DAIS

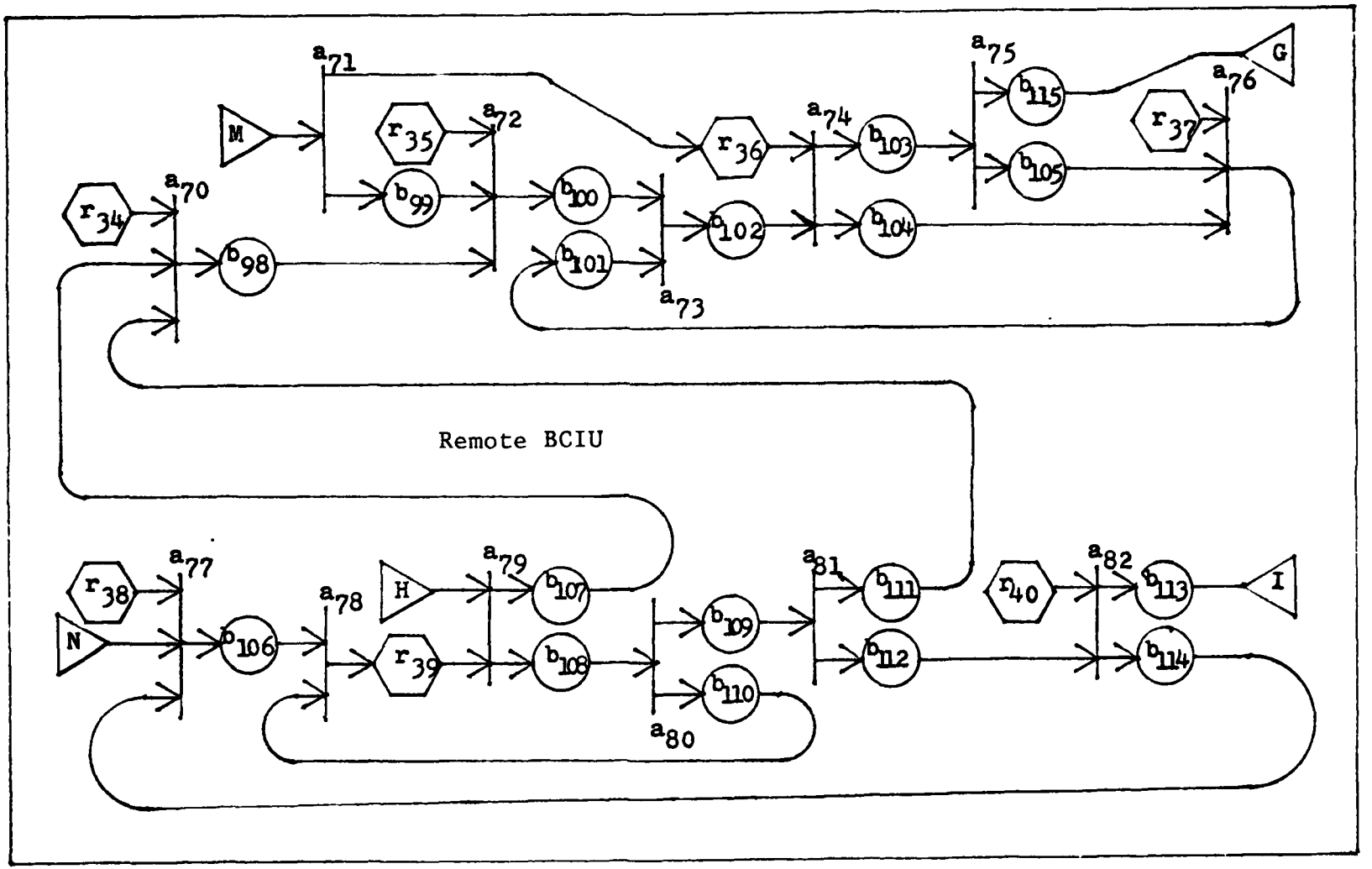


Fig. 18 (6 Of 6) Task Level E-Net Graph of the DAIS

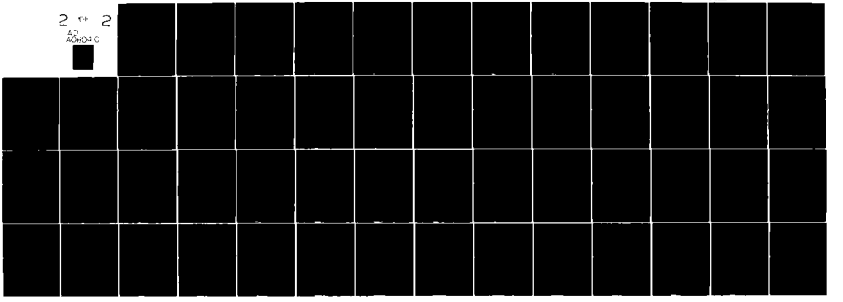
DAIS configuration modeled (Ref 24:28-32). Therefore, these functions are not incorporated into the task level E-net graph. Control and task flow required for system synchronization are incorporated into the model. The master processor sends commands to the remote processor via the subnet interconnections labeled D,E,H, and I in that order.

Data bus control is represented by the interconnections of the data bus and BCIUs and by the flow of tokens representing tasks and signals. The PIO connection between the master (remote) processor and the master (remote) BCIU is represented by location b_{38} (b_{96}). The master BCIU places commands on the bus at location b_{51} and receives responses at location b_{58}. The remote BCIU processes commands it receives from the bus at location b_{57} and places responses to commands on the bus at location b_{115}. Locations b_{52} and b_{55} represent the connection of the RTs and other bus-compatible avionics subsystems to the data bus.

A BCIU sends an interrupt to its processor whenever it receives a bus message which requires executive action. The master processor receives an interrupt whenever the master BCIU receives a "request for services" from the remote processor. The local processor receives an interrupt whenever its BCIU receives an asynchronous transmit or receive command. These interrupts are received by the master and remote processors at locations b_{50} and b_{113}, respectively. An interrupt is processed in the model by executing an interrupt task which is

AD-A080 410 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A GRAPH MODEL REPRESENTATION OF A DISTRIBUTED PROCESSOR COMPUTE--ETC(U)
DEC 79 L A PALMER

UNCLASSIFIED AFIT/GCS/EE/79-12 NL



END
DATE FILMED
3-80
DDC

created at locations b_6 and b_{64} for the master and remote processors, respectively.

Task control and the task priority scheme are interrelated. A task is represented by an attribute token, K[8]:

1) K(1) : 1 if an interrupt task, 0 otherwise
2) K(2) : 1 if a Normal Mode task, 0 otherwise
3) K(3) : task priority (1 is highest)
4) K(4) : processor time required, in milliseconds
5) K(5) : processor time used, in milliseconds
6) K(6) : 1 for synchronous bus command
   2 for asynchronous bus command
7) K(7) : integer value representing a request for services
8) K(8) : BCIU processing time.

The position of a task (token) in a processor priority-in queue, $Q_m[8]$ or $Q_p[8]$, is determined by the value of the task's (token's) second attribute. All interrupt tasks have priority 1, all Priviledged Mode and executive tasks have priority 2, and Normal Mode tasks have priority 3 thru n, where n-2 is the number of Normal Mode tasks in the processor.

Task control is concerned with the order of execution of tasks, the allocation and deallocation of a processor to a task, and the thread of control from a task back through the calling sequence to the Master Sequencer (Figure 9) or the executive. The order of execution of tasks is controlled with the processor queues, $Q_m[8]$ and $Q_p[8]$. The allocation of the processors is controlled at r-locations r_5 and r_{26}. When the master (remote)

processor becomes idle (location b_{17} (b_{75}) is empty), the task at the head of Q_m(Q_p) is removed from the queue and receives control of the processor. The allocation of the master (remote) processor is represented by the successive firings of transitions a_{11} and a_{12} (a_{52} and a_{53}) after r (r_5) is set to 0.

Deallocation of the processors is more complicated.

Deallocation of a processor depends on the class and priority of the task currently in control of the processor (see page 33). Once a Privileged Mode or executive task starts executing it runs to completion. However, execution of a Normal Mode task is suspended whenever a higher priority task becomes executable. In the model, the master (remote) processor is deallocated by the successive firings of transitions a_{17} and a_{18} (a_{58} and a_{59}). The marking of r-location r_6 (r_{27}) is undefined until one of two things happens. If an interrupt is received by the master (remote) processor during execution of a Normal Mode task, then a token is placed on location b_8 (b_{66}), which will cause the marking of r_6 (r_{27}) to be changed to 1, enabling transition a_{17} (a_{58}). If a Privileged Mode or executive task is executing, or if an executing Normal Mode task is not suspended, then the normal termination of the executing task will define the value of $M(r_6)$(of $M(r_{27})$) to be 0. Transition a_{17} (a_{58}) then fires, since a token is always available on location b_{22} (b_{80}).

Processor activity is represented by a token on locations b_{17} and b_{75} for the master and remote processors, respectively. Tokens on these locations represent the execution of applications tasks, executive services, and interrupt processing. The other transitions and locations in the processor portions of the E-net model system overhead activities such as allocation of the processor, clearing of interrupts, entering tasks in the queues, and PIO. BCIU activity is represented by the allocation of the BCM in the master and remote BCIUs (locations b_{48} and b_{111}, respectively), the transmission and reception of messages (locations b_{51}, b_{57}, b_{58}, and b_{115}), and the generation of interrupts (locations b_{50} and b_{113}).

Interpretation of the E-net graph in Figure 18 simulates DAIS operation. It is assumed that the net environment, places attribute tokens (tasks) on locations b_{11} and b_{69} to simulate the activation of executable tasks in the DAIS. It is also assumed that the environment changes the value of the environment variable CLOCK to represent the passage of real time. Given an initial marking M_o for the graph, the model is executed by firing transitions as they are enabled and in accordance with the formal definition in Appendix C. An example is presented in the next section.

Interpretation of the model results in the automatic collection of some data. Locations b_3, b_9, b_{18}, b_{33}, b_{36}, b_{61}, b_{67}, b_{76}, b_{91}, and b_{94} are used to collect data on

interrupts and tasks (see Appendix C for location definitions). Other performance data can be calculated by measuring dwell times at locations b_{17}, b_{48}, b_{75}, b_{110}, Q_m, Q_n, and Q_p. In the next section, an example of how a token (task) flows through the graph (DAIS) is presented.

Model Validation. Validation of any model is generally achieved by convincing oneself that the model actually represents the structure and/or the function of the modeled system. For a simulation model, validation is accomplished by "exercising" the model with a "representative" (and, of course, validated) workload model. If the performance of the model approximates the actual system performance under the workload that was modeled, then the model is validated. How accurate the approximation must be for a particular validation effort is subjective.

For this investigation, validation of the high and intermediate level models was accomplished by exercising these models in a manner similar to the example at the bottom of page 71. It was only necessary to validate the structure of these two models since, for this investigation, they were only to be used as stepping stones to the task level model.

The task level model was initially validated in the same manner as the first two models. The structure of the task level model was validated in subsections, and then pieced together for an overall validation effort. Validation of the function of the model was accomplished by introducing tokens to the net at locations b_{11} and b_{69} and then interpreting

the model. An example of this effort is in Appendix D.

The bookkeeping involved during the graph interpretation becomes quite tedious and prone to error as the interpretation progresses. A more convincing validation process might involve an automated interpretation of the model with a workload model derived from an actual workload, but the amount of effort required is beyond the resource limitations of this investigation.

Summary

A set of guidelines was established as a basis for the model construction effort. A top down approach was used during model development. Three models were constructed; the third one represents the DAIS at a task flow level of detail. All three models were validated with respect to their structure. The task level model was interpreted to provide a rough validation of its function and performance. An evaluation of the three models constructed is provided in the next chapter.

IV Evaluation Of Models

E-nets were developed as a system analysis tool (Ref 15:23). Analysis of the net interpretation provides performance measures of the system represented by the net. Since E-nets are also pictoral representations of the systems they model, it should also be possible to perform some system analysis based on the uninterpreted graph representation of the modeled system. An evaluation of the models constructed during this investigation as analysis tools is presented in this chapter. Some possible methods of analyzing the graphs are also discussed.

Structural Analysis

All three models constructed could be used to analyze the DAIS architecture. The interconnection of system components is represented by the interconnection of nodes in the net. The three models can be viewed as block diagrams of the DAIS, at successively greater levels of detail. With the nodes defined and labeled, the control and data paths between system components are apparent. Referring to Figure 18, location b_{38} represents a register in the master BCIU which receives PIO instructions from the master processor. The master BCIU is interfaced to the data bus via the MTU which is represented by location b_{51}.

Part of the interface between the system and its environment is represented by the peripheral locations in the net

graphs. In Figure 18, locations b_{52} and b_{55} represent the connection of bus-compatible avionics subsystems to the avionics computer system (DAIS).

A high level representation of some DAIS system functions is also revealed in the graph structures. Also, the breakdown of some functions into subfunctions is revealed. In Figure 19, the overall system function is composed of the processor/BCIU and data bus functions. The master processor/BCIU function is composed of the processor allocation, processor deallocation, and BCIU function. (Figure 19 could also represent the physical composition of the system.)

With the nodes defined, the graphs represent the events or activities occurring in the system and the conditions which must be satisfied for the events to take place. It is apparent, using the definitions in Appendix C for nodes r_6, a_{17}, b_8, b_{22}, and b_{23}, from the graph in Figure 18 that the master processor will be deallocated whenever the currently executing task terminates normally, or if an interrupt occurs during execution of a Normal Mode task.

The graph structures can be used to analyze the DAIS structure and part of the interface between the DAIS and its environment. It is possible, given the node definitions, to analyze the ordering of sequences of events. In order to analyze the performance of the DAIS, however, it is necessary to interpret the nets.

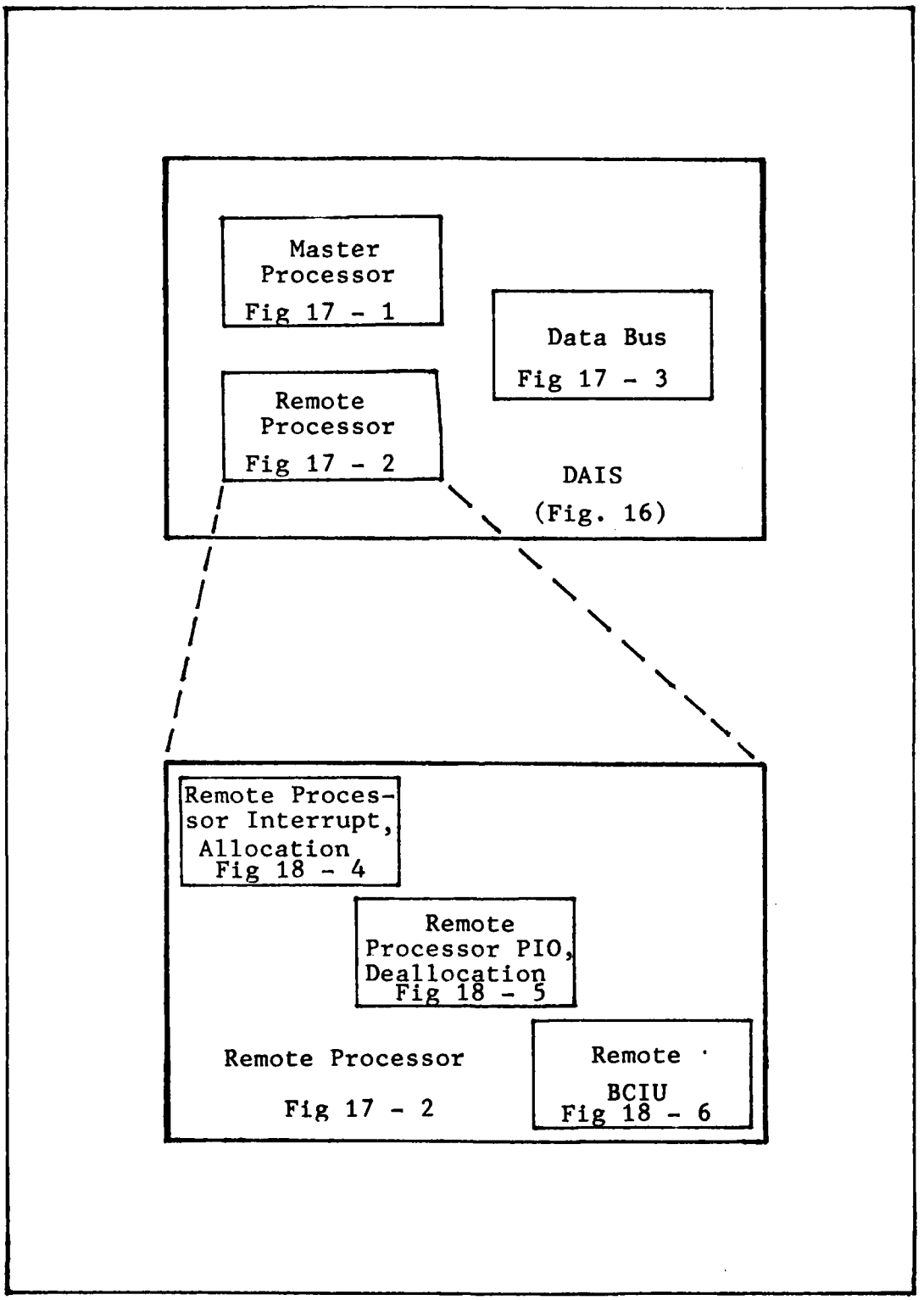


Fig. 19 Top Down Model Construction Overview

Dynamic Analysis

Examples of how to interpret E-nets are provided in Chapters two and three. The performance measures which could be made by interpreting each of the three models are discussed in Chapter three. Other performance-related issues could be studied by interpreting the nets.

If, during the interpretation of the task level net a marking was reached in which no transition was enabled (or would be enabled within a finite time period for the task level net), then the net would be dead. A dead net represents a hung-up or deadlocked system. If all interpretations of the task level net, initialized with the same M_o, CLOCK value, and task input times, resulted in the same terminal marking at time t_T, then the net (and the DAIS) is deterministic.

Interpretation of all three models developed is possible. However, sets of transition procedures and resolution procedures, and the location data structures of the high and intermediate level nets will have to be provided before these two nets could be interpreted. Interpretation of each net would provide performance measures at corresponding levels of detail. Several possible ways to interpret the nets are discussed in the next section.

Direct Interpretation. One method of interpreting the nets in Chapter three would be to assign an initial marking, initialize a subset of the net's environment variables, and then record the changes in the net marking as transitions become enabled and fire. An example of this method is in

Appendix D. Additionally, a data table could be constructed, where each entry in the table represents a marking of the net. Such a table (or similar method of recording data) would be necessary to monitor the net operation when several tokens are moving about in the net concurrently. For even a relatively simple net like the high level net, the recording effort can be substantial, and for a large, complex model, the recording effort can be prohibitive.

Data recording associated with an interpretation of the high level model could be done manually, although much time would be required to simulate DAIS operation of even a few minutes. An intermediate level net interpretation requires a substantially greater recording effort. If attribute tokens are used for an intermediate level net interpretation, then a vector of n values (where n is the number of token attributes) is associated with each token. A single row entry requires 41 columns, one for each location, in a table representing the net markings. Additionally, a table of token values must be maintained where each row entry corresponds to an attribute token in the net and each column entry corresponds to an attribute of the token. It is easily derived from the example in Appendix D and the discussion above that to manually record the data generated by a task level net interpretation is out of the question. An alternative method of collecting and recording the data associated with a task level net interpretation is required.

A net interpretation could be automated. A net interpretation computer program could be constructed as a main program with calls to subprograms. The main program would contain the logic necessary to sequence the subprogram calls. Subprograms would implement transition procedures. A subprogram call would only be valid if the transition was enabled. Data recording could be done in the main program, a common subprogram, or in each subprogram. Ideally, an automated interpretation of the models in Chapter three would be executed in real time on a multiple processor computer system with concurrent processing capabilities. However, not many systems like this are available. Also, the programming effort required for a direct interpretation program could be prohibitive or not cost-effective.

Simulation. An alternative to direct interpretation with a computer program is simulation of the net interpretation with a computer program. Several simulation programs exist which could conceivably be used (GASP-IV, GPSS, Q-GERT, and SLAM are a few examples). A brief discussion of how the GASP-IV program could be used to simulate interpretation of the task level net follows.

The main functions performed by GASP-IV are data collection and supervisory control of the simulation. Additional functions include statistical analyses of the data collected, generation of inputs to the model, and report generation. A GASP-IV model of a net in Chapter three could consist of a

set of queues and servers, representing locations and transitions, respectively.

GASP-IV supports event-and time-driven simulations. If enabling a transition is represented by an event, then an event-driven simulation of a net interpretation would be possible. GASP-IV provides and maintains up to ten files for recording events. Input to an event file would correspond to enabling a transition. Output from the same event file would correspond to termination of a transition firing. Event file entries can have several components, one of which is used to place the entry in the file. The file above could be ordered by transition firing times.

All simulation programs have implementation bounds on storage for data collection and recording and on the size of the simulation model. These restrictions may limit the use of some simulation programs for net interpretation.

Summary

E-nets were developed as a performance analysis tool. It is possible to analyze the structure and functional relationships of the DAIS by analyzing the nets in Chapter three. A performance analysis requires an interpretation of the nets. Although it would be possible to manually interpret the net models of the DAIS, the data collection and recording effort would be substantial even for the high level net. Possible alternatives to manual interpretation of the nets are automated direct interpretation and simulation.

V Results And Conclusions

In this chapter, the results of this investigation are presented, followed by some concluding remarks. Then some recommendations for future work are offered.

Results

The physical end product of this investigation is a set of three E-net graph models of the DAIS, a distributed processor avionics computer system architecture. The three models represent the DAIS at successively lower levels of detail. The highest level net models the major components (e.g. processor) and the lowest level net represents the DAIS at the task flow level of observation.

It is demonstrated that a particular class of graph models, E-nets, can be used to model a distributed computer architecture. Several methods for analyzing the models constructed are presented, including analysis of the static graph structures and analysis of interpreted nets (dynamic graphs).

It is shown how, by using tokens (markers) in the net graph to represent jobs (tasks) in the system model, to simulate DAIS operation. Methods for collecting performance data during a simulation (net interpretation) are presented.

An evaluation of the models constructed is presented. The evaluation discusses how the models could be used to analyze the structural, functional, and operational

characteristics of the DAIS. The problem of collecting and recording data during a DAIS simulation is discussed and examples are provided.

Conclusions

The top down approach used to construct the E-net graph models of this investigation was quite effective. The three models produced provide three separate levels of observation of the DAIS. The inherent modularity of E-nets aided the construction effort, reducing the difficulty of interconnecting subsections of a net graph as they were developed. A restriction of E-nets is that the termination time of a net activity (event) during net interpretation is fixed when the activity is initiated. Consequently, the processor interrupt function of the DAIS is not obvious in the graph and required the only use of net environment variables when the model was executed.

The parallel processing capability is incorporated into the net interpretation and the distributed architecture of the DAIS is represented in the graph structure. Although the interpretation of E-nets is straightforward, the bookkeeping involved during interpretation of even a relatively simple net can be substantial and tedious. Interpretation of any of the E-net graphs constructed during this investigation should be facilitated by data automation techniques.

Recommendations for Future Work

The product of this investigation is a set of E-net graphs which represent the DAIS computer system. The models

can be used to analyze the structure of and the functional relationships in the DAIS. However, to analyze the performance and operational characteristics of the DAIS, the net graphs need to be interpreted (the model is executed). Workload models need to be developed for input to the DAIS models.

In addition to a workload model, a data automated capability needs to be developed to perform the data recording function associated with a net interpretation. Preferably, the entire interpretation, including the bookkeeping function, could be automated. Two alternatives for automating the interpretation of the net graphs are: a) programs could be written which would interpret the graphs directly, or b) an existing simulation program could be used to simulate the net interpretations.

Bibliography

1. AFAL-TR-74-245. Design of the Core Elements of the Digital Avionics Information System (DAIS), Volume I. Product Report. Dallas: Texas Instruments Incorporated, November, 1974.

2. AFAL-TR-74-245. Design of the Core Elements of the Digital Avionics Information System (DAIS), VolumeII. Product Report. Dallas: Texas Instruments Incorporated, November, 1974.

3. AFAL-TR-79-1027. Digital Avionics Information System (DAIS): Development and Demonstration. Technical Report. Redondo Beach, California: TRW Defense and Space Systems Group, Inc., March 1979.

4. Ferrari, Domenico. Computer Systems Performance Evaluation. Englewood Cliffs: Prentice-Hall, Inc., 1978.

5. Gordon, Geoffrey. System Simulation (Second edition). Englewood Cliffs: Prentice-Hall, Inc., 1978.

6. Hetzel, William C. Program Test Methods. Englewood Cliffs: Prentice-Hall, Inc., 1973.

7. Lamont, Gary B. "Graphical and Functional Models of Parallel Computation." Report distributed in EE 7.50, Advanced Operating Systems. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1978.

8. MA100100. DAIS Document Descriptions: Specifications, Manuals, Plans and Drawings. Wright-Patterson AFB: DAIS ADPO, August, 1978.

9. MA201200. DAIS System Control Procedures. Wright-Patterson AFB: DAIS ADPO, April 1979.

10. MA301300. DAIS Technical Manual For the Bus Control Interface Unit (BCIU). Wright-Patterson AFB: DAIS ADPO, April, 1977.

11. MA301301. DAIS Technical Manual For Remote Terminals. Wright-Patterson AFB: DAIS ADPO, April, 1977.

12. Martin, Francis F. Computer Modeling and Simulation. New York: John Wiley & Sons, Inc., 1968.

13. Misunas, D. "Petri Nets and Speed Independent Design," Communications of the ACM, 16: 474-481 (August, 1973).

14. Nutt, Gary J. Evaluation Net Simulation System Reference Manual. Department of Computer Science, University of Colorado, Boulder, Colorado, April, 1974.

15. Nutt, Gary J. "The Formulation and Application of Evaluation Nets." Ph.D. Thesis. Computer Sciences Group, University of Washington, Seattle, Washington, July, 1972.

16. Nutt, Gary J. and Jerry Noe. "Macro E-Nets for Representation of Parallel Systems," IEEE Transactions on Computers, C-22: 718-727 (August, 1973).

17. Nurr, Gary J. and Jerry Noe. "Some evaluation Net Macro Structures," Computer Sciences Group, University of Washington, Seattle, Washington, TR 73-01-07, 1973.

18. PA100101. Technical Description of the Digital Avionics Information System (DAIS). Wright-Patterson AFB: DAIS ADPO, February, 1978.

19. PA100203A. Demonstration and Acceptance Test Plan For Digital Avionics Information System Missions α,β,γ, and δ. Wright-Patterson AFB: DAIS ADPO, February, 1979.

20. PA200100. DAIS Software Development Standards. Wright-Patterson AFB: DAIS ADPO, February, 1978.

21. Peterson, James L. "Petri Nets," Computing Surveys, 9: 223-252 (September 1977).

22. Ramchandani, Chander. "Analysis of Asynchronous Concurrent Systems by Petri Nets." Ph.D. Thesis. Department of Electrical Engineering, M.I.T., Cambridge, Massachusetts, February, 1974.

23. SA100100A. System Specification for the Digital Avionics Information System. Wright-Patterson AFB: DAIS ADPO, July, 1977.

24. SA100102A. System Segment Specification for the Digital Avionics Information System Missions α,β,γ, and δ Type A. Wright-Patterson AFB: DAIS ADPO, December, 1978.

25. SA201302 Pt I. DAIS Mission Software Executive Specification. Wright-Patterson AFB: DAIS ADPO, June, 1976.

26. SA201302 Pt II, Vol I. DAIS Mission Software Product Specification: Local Executive. Wright-Patterson AFB: DAIS ADPO, August, 1976.

27. SA201302 Pt II, Vol II. DAIS Mission Software Product Specification: Bus Control Executive. Wright-Patterson AFB: DAIS ADPO, March, 1979.

28. SA201302 Pt II, Vol III. DAIS Mission Software Product Specification: Start Up/Loader Mass Memory Controller. Wright-Patterson AFB: DAIS ADPO, unpublished.

29. SA201302 Pt II, Vol IV. DAIS Mission Software Product Specification: Monitor, Back-Up, Recovery Reconfiguration. Wright-Patterson AFB: DAIS ADPO, unpublished.

30. SA301300B. Prime Item Development Specification For DAIS Bus Control Interface Unit. Wright-Patterson AFB: DAIS ADPO, March, 1976.

31. SA321200. Prime Item Development Specification For DAIS Digital, Command/Response, Time Division Multiplexing Data Bus. Wright-Patterson AFB: DAIS ADPO, February, 1979.

32. Svobodova, Liba. Computer Performance Measurement and Evaluation Methods: Analysis and Applications. New York: American Elsevier Publishing Company, Inc., 1978.

33. Vandever, Woodrow H., Jr. "The DAIS Executive: An Introduction," Unpublished report. Intermetrics, Incorporated, Dayton Facility, Dayton, Ohio, 1978.

34. Zervos, Cristian Radu and Keki B. Irani. "Colored Petri Nets: Their Properties and Applications," Interim report. Department of Electrical Engineering, University of Michigan, Ann Arbor, Michigan, August, 1977.

Appendix A: BNF Notation

The Backus-Naur-Form (BNF) notation used to describe a transition procedure (Ref 15:65-66) is described using the following example:

<macro> ::= <micro> | <macro><op><micro>

<micro> ::= <basic> | <basic><micro>

<basic> ::= a | b | c

<op> ::= + | *

The constructs <macro>,<micro>,<op>, and <basic> are non-terminal symbols. The symbols a,b,c,+, and * are terminal symbols. The symbols | and ::= are meta symbols. The symbol ::= is interpreted to mean "replace with". Whatever is on the left side of ::= may be replaced by whatever is on the right side. The symbol | indicates a choice or option. That is, <op> may be replaced by + or * . Using the rules for replacement illustrated in the above example, the following are legal constructions made up of terminal symbols:

ab + ab

aba * bcb + cac .

For the notation used in the transition procedure definition, ::= and | are meta symbols, anything enclosed with (and including for notation purposes) the symbol pair < > is a non-terminal symbol, and all other symbols are terminal symbols.

Appendix B Definition of the Intermediate Level E-Net Graph of the DAIS

The nodes in the E-net graph of Figure 17 are defined as follows:

a_1: pass activated or suspended job to remote processor queue

a_2: Add job or asynchronous request to master processor queue

a_3: allocate the master processor

a_4: deallocate the master processor

a_5: pass bus messages to master BCIU queue

a_6: feed suspended jobs back to master processor queue

a_7: add bus messages and processor commands to master BCIU queue

a_8: allocate BCM of master BCIU

a_9: deallocate BCM of master BCIU

a_{10}: route messages for transmission to data bus

a_{11}: route asynchronous requests to master processor

a_{12}: pass activated job or suspended job to remote processor queue

a_{13}: add job or asynchronous command to remote processor queue

a_{14}: allocate the remote processor

a_{15}: deallocate the remote processor

a_{16}: pass bus messages to the remote BCIU queue

a_{17}: feed suspended jobs back to remote processor queue

a_{18}: add bus messages and processor commands to remote BCIU queue

a_{19}: Allocate BCM of remote BCIU

a_{20}: deallocate BCM of remote BCIU

a_{21}: route messages for transmission to data bus

a_{22}: route asynchronous commands to remote processor

a_{23}: pass bus messages from processors to bus

a_{24}: pass RT transmissions and processor messages to data bus

a_{25}: allocate the data bus

a_{26}: deallocate the data bus

a_{27}: route messages to RT or processor/BCIU

a_{28}: route messages to master or remote BCIU

b_1 : a job is ready to execute

b_2 : an executable job requests the master processor

b_3 : master processor is allocated

b_4 : master processor is deallocated

b_5 : job execution terminated

b_6 : a terminated job did not generate a bus message

b_7 : bus message needs to be transmitted

b_8 : terminated job was suspended

b_9 : terminated job completed execution

b_{10}: BCM of master BCIU allocated

b_{11}: BCM of master BCIU deallocated

b_{12}: message needs to be routed

b_{13}: message was received from bus

b_{14}: message is ready for transmission

b_{15}: asynchronous request needs to be processed

b_{16}: received message requires no further processing

b_{17}: a job is ready to execute

b_{18}: an executable job requests the remote processor

b_{19}: remote processor is allocated

b_{20}: remote processor is deallocated

b_{21}: job execution terminated

b_{22}: terminated job did not generate a bus message

b_{23}: bus message needs to be transmitted

b_{24}: terminated job was suspended

b_{25}: terminated job completed execution

b_{26}: BCM of remote BCIU allocated

b_{27}: BCM of remote BCIU is deallocated

b_{28}: message needs to be routed

b_{29}: message was received from bus

b_{30}: message is ready for transmission

b_{31}: asynchronous command needs to be processed

b_{32}: received message requires no further action

b_{33}: message from RT

b_{34}: message from a processor

b_{35}: data bus requested

b_{36}: data bus busy

b_{37}: data bus idle

b_{38}: message transmission complete

b_{39}: message is for a processor

b_{40}: message to RT

b_{41}: message is to master processor

b_{42}: message is to remote processor

Q_m : master processor queue

Q_n : master BCIU queue

Q_p : remote processor queue

Q_s : remote BCIU queue

r_1 : choose jobs for master processor

r_2 : choose jobs and requests for input to master processor queue

r_3 : detect bus messages

r_4 : detect job complete or suspended

r_5 : choose bus messages for master BCIU queue

r_6 : detect messages to be transmitted

r_7 : determine if message is asynchronous request

r_8 : choose jobs for remote processor

r_9 : choose jobs and requests for input to remote processor queue

r_{10}: detect bus messages

r_{11}: detect job complete or suspended

r_{12}: choose bus messages for remote BCIU queue

r_{13}: detect messages to be transmitted

r_{14}: determine if message is asynchronous command

r_{15}: choose processor messages to be placed on bus

r_{16}: choose processor and RT messages to be placed on bus

r_{17}: detect messages to RT

r_{18}: determine if message is to master or remote processor

The formal definition of the net in Figure 17 is:

$E = (L,P,R,A)$

$R = \{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8,r_9,r_{10},r_{11},r_{12},r_{13},r_{14},r_{15},$
$r_{16},r_{17},r_{18}\}$

$P = \{b_1,b_9,b_{16},b_{17},b_{25},b_{32},b_{33},b_{40}\} \cup R$

$L = \{b_2,\ldots,b_8,b_{10},\ldots,b_{15},b_{18},\ldots,b_{24},b_{26},\ldots,b_{31},$
$b_{34},\ldots,b_{40}\} \cup P$

$A = \{a_1,a_2,\ldots,a_{28}\}$

$a_1 = (Y(r_1,b_1,b_8,b_2) , (t_0(a_1),t_1(a_1)))$

$a_2 = (Y(r_2,b_2,b_{15},Q_m) , (t_0(a_2),t_1(a_2)))$

$a_3 = (J(Q_m,b_4,b_3) , t(a_3))$

$a_4 = (F(b_3,b_5,b_4) , t(a_4))$

$a_5 = (X(r_3,b_5,b_6,b_7) , (t_0(a_5) , t_1(a_5)))$

$a_6 = (X(r_4,b_6,b_8,b_9) , (t_0(a_6), t_1(a_6)))$

$a_7 = (Y(r_5,b_7,b_{41},Q_n) , (t_0(a_7) , t_1(a_7)))$

$a_8 = (J(Q_n,b_{11},b_{10}) , t(a_8))$

$a_9 = (F(b_{10},b_{12},b_{11}) , t(a_9))$

$a_{10} = (X(r_6,b_{12},b_{13},b_{14}) , (t_0(a_{10}) , t_1(a_{10})))$

$a_{11} = (X(r_7,b_{13},b_{15},b_{16}) , (t_0(a_{11}) , t_1(a_{11})))$

$a_{12} = (Y(r_8,b_{17},b_{24},b_{18}) , (t_0(a_{12}) , t_1(a_{12})))$

$a_{13} = (Y(r_9,b_{18},b_{31},Q_p) , (t_0(a_{13}) , t_1(a_{13})))$

$a_{14} = (J(Q_p, b_{20}, b_{19}) , t(a_{14}))$

$a_{15} = (F(b_{19}, b_{21}, b_{20}) , t(a_{15}))$

$a_{16} = (X(r_{10}, b_{21}, b_{22}, b_{23}) , (t_0(a_{16}), t_1(a_{16})))$

$a_{17} = (X(r_{11}, b_{22}, b_{24}, b_{25}) , (t_0(a_{17}) , t_1(a_{17})))$

$a_{18} = (Y(r_{12}, b_{23}, b_{42}, Q_s) , (t_0(a_{18}) , t_1(a_{18})))$

$a_{19} = (J(Q_s, b_{27}, b_{26}) , t(a_{19}))$

$a_{20} = (F(b_{26}, b_{28}, b_{27}) , t(a_{20}))$

$a_{21} = (X (r_{13}, b_{28}, b_{29}, b_{30}) , (t_0(a_{21}) , t_1(a_{21})))$

$a_{22} = (X(r_{14}, b_{29}, b_{31}, b_{32}) , (t_0(a_{22}) , t_1(a_{22})))$

$a_{23} = (Y(r_{15}, b_{14}, b_{30}, b_{34}) , (t_0(a_{23}) , t_1(a_{23})))$

$a_{24} = (Y(r_{16}, b_{33}, b_{34}, b_{35}) , (t_0(a_{24}) , t_1(a_{24})))$

$a_{25} = (J(b_{35}, b_{37}, b_{36}) , t(a_{25}))$

$a_{26} = (F(b_{36}, b_{38}, b_{37}) , t(a_{26}))$

$a_{27} = (X(r_{17}, b_{38}, b_{39}, b_{40}) , (t_0(a_{27}) , t_1(a_{27})))$

$a_{28} = (X(r_{18}, b_{39}, b_{41}, b_{42}) , (t_0(a_{28}) , t_1(a_{28})))$

Appendix C Definition of the Task Level E-Net Graph of the DAIS

The definitions of the nodes in the E-net graph in Figure 18 are as follows:

a_1 : route master processor interrupts

a_2 : collect interrupt flags

a_3 : pass interrupts to master processor queue

a_4 : absorb interrupt flags

a_5 : route interrupt flags

a_6 : collect interrupt falgs which occur during execution of non-Normal Mode tasks

a_7 : absorb interupt flags which occur during execution of non-Normal Mode tasks

a_8 : pass activated tasks and suspended Normal Mode tasks to the master processor queue

a_9 : pass executable tasks to master processor queue

a_{10}: add interrupts and tasks to master processor queue

a_{11}: remove task from head of master processor queue

a_{12}: allocate the master processor

a_{13}: project time at which currently executing task would end if it runs to completion

a_{14}: absorb end-of-task flags

a_{15}: pass end-of-task flags

a_{16}: create end-of-task flags

a_{17}: pass end-of-task flags and interrupts

a_{18}: deallocate the master processor

a_{19}: separate Normal Mode tasks from other tasks

a_{20}: route suspended Normal Mode tasks back to master processor queue

a_{21}: separate BCIU related tasks from other non-Normal Mode tasks

a_{22}: separate BCIU related tasks from other Normal Mode tasks

a_{23}: collect non-BCIU related non-Normal Mode tasks

a_{24}: absorb non-BCIU related non-Normal Mode tasks

a_{25}: collect non-BCIU related, completed Normal Mode tasks

a_{26}: absorb non-BCIU related, completed Normal Mode tasks

a_{27}: collect BCIU related tasks

a_{28}: route BCIU tasks to the master BCIU queue

a_{29}: route messages from bus to BCM in master BCIU

a_{30}: collect tokens signalling end of BCM activity

a_{31}: pass messages from bus to BCM in master BCIU

a_{32}: collect end-of-action flags for BCM in master BCIU

a_{33}: un-busy the BCM in the master BCIU

$_{34}$: add BCIU operations to master BCIU queue

a_{35}: allocate the BCM in the master BCIU

a_{36}: process the BCIU operation

a_{37}: route outgoing messages to bus, incoming asynchronous requests to the master processor

a_{38}: collect bus messages from remote BCIU and RT

a_{39}: messages placed on the bus

a_{40}: route messages off bus

a_{41}: route messages to BCIUs

a_{42}: route remote processor interrupts

a_{43}: collect interrupt flags

a_{44}: pass interrupts to remote processor

a_{45}: absorb interrupt flags

a_{46}: route interrupt flags

a_{47}: collect interrupt flags which occur during execution of non-Normal Mode tasks

a_{48}: absorb interrupt flags which occur during execution of non-Normal Mode tasks

a_{49}: pass activated tasks and suspended Normal Mode tasks to the remote processor queue

a_{50}: pass executable tasks to remote processor queue

a_{51}: add interrupts and tasks to remote processor queue

a_{52}: remove task from head of remote processor queue

a_{53}: allocate the remote processor

a_{54}: project time at which currently executing task would end if it runs to completion

a_{55}: absorb end-of-task flags

a_{56}: pass end-of-task flags

a_{57}: create end-of-task flags

a_{58}: pass end-of-task flags and interrupts

a_{59}: deallocate the remote processor

a_{60}: separate Normal Mode tasks from other tasks

a_{61}: route suspended Normal Mode tasks back to remote processor queue

a_{62}: separate BCIU related tasks from other non-Normal Mode tasks

a_{63}: separate BCIU related tasks from other Normal Mode tasks

a_{64}: separate BCIU related tasks from other Normal Mode tasks

a_{65}: absorb non-BCIU related non-Normal Mode tasks

a_{66}: collect non-BCIU related, completed Normal Mode tasks

a_{67}: absorb non-BCIU related, completed Normal Mode tasks

a_{68}: collect BCIU related tasks

a_{69}: route BCIU related tasks to the remote BCIU queue

a_{70}: pass response or message to response

a_{71}: pass remote processor request

a_{72}: pass remote processor request or remote BCIU response

a_{73}: update remote processor/BCIU status

a_{74}: pass updated status or response + status

a_{75}: pass response + status to bus and clear status in remote BCIU

a_{76}: pass current status

a_{77}: collect end of action flags for the BCM in the remote BCIU

a_{78}: un-busy the BCM in the remote BCIU

a_{79}: pass BCIU tasks to BCM when BCM not busy

a_{80}: allocate the BCM in the remote BCIU

a_{81}: route messages to bus

a_{82}: pass asynchronous commands to remote processor

b_1 : interrupt received

b_2 : an interrupt needs to be processed

b_3 : current count of interrupts

b_4 : updated count of interrupts

b_5 : interrupt flag

b_6 : task to process an interrupt

b_7 : interrupt occurred during execution of Priviledged Mode or executive task

b_8 : interrupt occurred during execution of a Normal Mode task

b_9 : current count on interrupts when executing task not a Normal Mode task

b_{10}: updated count of interrupts when executing task not a Normal Mode task

b_{11}: activated task ready to execute

b_{12}: activated task or suspended task requests master processor

b_{13}: executable task requests master processor

b_{14}: task at head of queue bumped by entry into queue of higher priority task

b_{15}: highest priority task in queue is selected for execution

b_{16}: copy of task in master processor

b_{17}: master processor allocated

b_{18}: current count of allocation of the master processor

b_{19}: updated count of allocation of the master processor

b_{20}: updated count of completed tasks in master processor

b_{21}: current count of completed tasks in master processor

b_{22}: enabling token used when executing task runs to completion

b_{23}: stop execution of executing task

b_{24}: task just executed by master processor

b_{25}: completed Priviledged Mode or executive task

b_{26}: Normal Mode task was executing

b_{27}: Normal Mode task ran to completion

b_{28}: Normal Mode task was suspended

b_{29}: task just completed was not BCIU related

b_{30}: task just completed was BCIU related

b_{31}: Normal Mode task just completed generated a BCIU operation

b_{32}: Normal Mode task just completed did not generate a BCIU operation

b_{33}: current count of completed, non-BCIU related, Priviledged Mode and executive tasks in the master processor

b_{34}: updated count of completed, non-BCIU related, Priviledged Mode and executive tasks in the master processor

b_{35}: updated count of completed, non-BCIU related, Normal Mode tasks in the master processor

b_{36}: current count of completed, non-BCIU related, Normal Mode tasks in the master processor

b_{37}: BCIU operation needs to be processed

b_{38}: a bus message needs to be processed

b_{39}: interrupt has been processed by the master processor and BCM in master BCIU can be unbusied

b_{40}: status response received from remote device does not require processing

b_{41}: status response received from remote device requires further processing

b_{42}: status response received or master processor interrupt has been handled so BCM can be unbusied

b_{43}: response to last command is received

b_{44}: service request from remote device

b_{45}: the BCM in the master BCIU can be unbusied

b_{46}: the BCM in the master BCIU is not busy

b_{47}: a master BCIU operation requires processing

b_{48}: the BCM in the master BCIU is busy

b_{49}: a master BCIU operation is executing

b_{50}: a remote device response requires processing by the master processor

b_{51}: a message needs to be transmitted on bus by the master BCIU

b_{52}: an RT or the remote processor needs to be transmitted on the bus

b_{53}: a message needs to be transmitted on the bus

b_{54}: a message is on the bus

b_{55}: the message is for an RT

b_{56}: the message is for a processor

b_{57}: the message is for the remote processor

b_{58}: the message is for the master processor

b_{59}: interrupt received

b_{60}: an interrupt needs to be processed

b_{61}: current count of interrupts

b_{62}: updated count of interrupts

b_{63}: interrupt flag

b_{64}: task to process an interrupt

b_{65}: non-transmit interrupt occurred during execution of Priviledged mode or executive task

b_{66}: interrupt occurred during execution of a Normal Mode task

b_{67}: current count of interrupts during execution of a non-Normal Mode task

b_{68}: updated count of interrupts during execution of a non-Normal Mode task

b_{69}: activated task ready to execute

b_{70}: activated task or suspended task requests remote processor

b_{71}: executable task requests remote processor

b_{72}: task at head of queue bumped by entry into queue of higher priority task

b_{94}: current count of completed, non-BCIU related, Normal Mode tasks in the remote processor

b_{95}: BCIU operation needs to be processed

b_{96}: a bus message needs to be processed

b_{97}: interrupt has been processed by the remote processor and BCM in the remote BCIU can be unbusied

b_{98}: signal that command was received or a message

b_{99}: remote processor request or flag

b_{100}: status update or message

b_{101}: current status

b_{102}: updated status or message + status

b_{103}: status response or message + status

b_{104}: updated status

b_{105}: cleared status

b_{106}: end of BCM activity or interrupt has been processed by remote processor, so BCM in the remote BCIU can be unbusied

b_{107}: acknowledge receipt of command from bus if BCIU is busy

b_{108}: a command received from bus needs to be processed

b_{109}: copy of message being processed by BCM in the remote BCIU

b_{110}: the BCM in the remote BCIU is busy

b_{111}: message is ready to be transmitted

b_{112}: bus command has been processed

b_{113}: the remote processor needs to take action in response to the received bus command

b_{114}: the received command requires no further action

b_{115}: status response or message + status

$Q_m[j]$: priority-in queue for master processor

$Q_n[j]$: queue for master BCIU

$Q_p[j]$: priority-in queue for remote processor

r_1 : detect if executing task is a Normal Mode task

r_2 : choose activated task or suspended task

r_3 : choose executable task

r_4 : choose task for input to master processor queue

r_5 : detect master processor idle or arrival to the queue of a task with a higher priority than the task at the head of the queue

r_6 : detect normal termination of executing task or detect an interrupt during execution of a Normal Mode task

r_7 : detect if task just executed was Normal Mode

r_8 : detect if Normal Mode task execution was suspended

r_9 : detect if executed Priviledged Mode or executive task generated a BCIU operation

r_{10}: detect if executed Normal Mode task generated a BCIU operation

r_{11}: choose master BCIU operations

r_{12}: detect if the master BCIU operation is to unbusy the BCM subsequent to processing a master processor interrupt

r_{13}: detect a request for service

r_{14}: choose completion of interrupt processing or status response with a request for service

r_{15}: choose response to unbusy the BCM

r_{16}: choose master processor generated or bus generated inputs to BCIU queue

r_{17}: detect activity requiring the master processor

r_{18}: choose RT or remote processor transmission onto data bus

r_{19}: choose messages for transmission on data bus

r_{20}: route messages to RT

r_{21}: route messages to remote or master processor

r_{22}: detect if executing task is a Normal Mode task

r_{23}: choose activated task or suspended task

r_{24}: choose executable task

r_{25}: choose task for input to remote processor queue

r_{26}: detect remote processor idle or arrival to the queue of a task with a higher priority than the task at the head of the queue

r_{27}: detect normal termination of executing task or an interrupt during execution of a Normal Mode task

r_{28}: detect if task just executed was Normal Mode

r_{29}: detect if Normal Mode task execution was suspended

r_{30}: detect if executed Priviledged Mode or executive task generated a BCIU operation

r_{31}: detect if executed Normal Mode task generated a BCIU operation

r_{32}: choose remote BCIU operations

r_{33}: detect if the remote BCIU operation is to unbusy the BCM subsequent to processing a remote processor interrupt

r_{34}: choose message + response or response

r_{35}: choose remote BCIU response or remote processor request

r_{36}: detect updated status or response

r_{37}: choose cleared status or updated status

r_{38}: detect end of message processing or end of interrupt processing by the remote processor

r_{39}: detect if BCM in remote BCIU is busy

r_{40}: detect if an input from the bus requires further processing by the remote processor

Some shorthand notation is used in the formal definition of the task level E-net graph. The symbol - in a transition declaration is used whenever the function of the transition is as follows:

1) T - transition: The transition firing moves the unaltered token on the input location to the output location.

2) F - transition: When the transition fired, identical copies of the token on the input location are placed on both output locations.

3) J - transition: When the transition fires, the attributes of the token placed on the output location are obtained by summing the corresponding attributes of the tokens on the two input locations.

4) X - transition: When the transition fires, the token placed on the selected output location is the token from the input location.

5) Y - transition: When the transition fires, the token placed on the output location is the token from the selected input location.

When an attribute token K[m] is transferred from location $b_i[m]$ to $b_j[m]$ without changing the values of any attribute, the transfer is denoted

$$M(b_j[m]) := M(b_i[m]).$$

When any attribute values are changed during the transfer (as specified by the transition procedure), the transfer is denoted

$$M(b_j(n)) := M(b_i(n)) \quad , \quad 1 \leq n \leq m$$

for each token attribute value changed. Only the attribute

values which are changed are individually notated in the transition declaration. T is the shorthand notation for the boolean value "true" when T appears in a transition procedure. Time is in milliseconds.

The formal definition of the E-net in Figure 18 is:

$E = (L,P,R,A)$

$R = \{r_1, r_2, \ldots, r_{40}\}$

$P = \{b_{11}[1],\ b_{52}[8],\ b_{55}[8],\ b_{69}\} \cup R$

$L = \{b_1[1],\ b_2[8],\ b_3[1],\ b_4[1],\ b_5,\ b_6[8],\ b_7,\ b_8,\ b_9[1],$
$b_{10}[1],\ b_{11}[8], \ldots,\ b_{17}[8],\ b_{18}, \ldots,\ b_{23},\ b_{24}[8], \ldots,$
$b_{38}[8],\ b_{39},\ b_{40},\ b_{41}[8],\ b_{42},\ b_{43},\ b_{44}[8],\ b_{45},\ b_{46},$
$b_{47}[8],\ b_{48},\ b_{49}[8], \ldots,\ b_{58}[8],\ b_{59}[1],\ b_{60}[8],$
$b_{61}[1],\ b_{62}[1],\ b_{63},\ b_{64}[8],\ b_{65},\ b_{66},\ b_{67}[1],\ b_{68}[1],$
$b_{69}[8], \ldots,\ b_{75}[8],\ b_{76}, \ldots,\ b_{81},\ b_{82}[8], \ldots,\ b_{96}[8],$
$b_{97},\ b_{98}[8], \ldots,\ b_{105}[8],\ b_{106},\ b_{107}[8],\ b_{108}[8],$
$b_{109}[8],\ b_{110},\ b_{111}[8],\ b_{112}[8],\ b_{113}[8],\ b_{114}\} \cup P$

$A = \{a_1, a_2, \ldots, a_{82}\}$

$a_1 = (F(b_{50}[8],\ b_1[1],\ b_2[8]),\ 0,$
$[T \rightarrow (M(b_2[8]) := M(b_{50}[8]);$
$M(b_1[1]) := 1_j)])$

$a_2 = (J(b_3[1],\ b_1[1],\ b_4[1]),\ 0,\ -)$

a_3 = (F(b_2[8], b_5, b_6[8]), 0,

[T → (M(b_5) := 1;

M(b_6[8] := M(b_2[8]))])

a_4 = (T(b_4[1], b_3[1]), 0, -)

a_5 = (X(r_1, b_5, b_7, b_8) , (0,0) , -)

a_6 = (J(b_9[1], b_7, b_{10}[1]) , 0,

[T → (M(b_{10}(a)) := M(b_9(1)) + 1)])

a_7 = (T(b_{10}[1], b_9[1]) , 0, -)

a_8 = (Y(r_2, b_{11}[8], b_{28}[8], b_{12}[8]) , (0,0) , -)

a_9 = (Y(r_3, b_{12}[8], b_{14}[8], b_{13}[8]) , (0,0) , -)

a_{10} = (Y(r_4, b_6[8], b_{13}[8], Q_m[8]) , (0,0) , -)

a_{11} = (X(r_5, Q_m[8], b_{15}[8], b_{14}[8]) , (0,0) , -)

a_{12} = (F(b_{15}[8], b_{16}[8], b_{17}[8]) , 0, -)

a_{13} = (J(b_{18}, b_{16}[8], b_{19}) , 0,

[T → (M(b_{19}) := 1;

END TASK(1) := CLOCK + M(b_{16}(4)))])

a_{14} = (T(b_{19}, b_{18}) , 0, -)

a_{15} = (T(b_{20}, b_{21}) , 0, -)

a_{16} = (F(b_{21}, b_{20}, b_{22}) , 0, -)

a_{17} = (Y(r_6, b_{22}, b_8, b_{23}) , (0,0) , -)

a_{18} = (J(b_{23}, b_{17}[8], b_{24}[8]) , (0,0) ,

[(END TASK(1) > CLOCK) →

($M(b_{24}(4))$:= END TASK(1) - CLOCK;

$M(b_{24}(5))$:= $M(b_{17}(5))$ + $M(b_{17}(4))$

$-M(b_{24}(4))$;

$M(b_{24}(i))$:= $M(b_{17}(i))$,

i = 1,2,3,6,7,8):

T → ($M(b_{24}(5))$:= $M(b_{17}(4))$ + $M(b_{17}(5))$;

$M(b_{24}(4))$:= 0;

$M(b_{24}(i))$:= $M(b_{17}(i))$,

i = 1,2,3,6,7,8)])

a_{19} = (X(r_7, b_{24}[8], b_{25}[8], b_{26}[8]) , (0,0) , -)

a_{20} = (X(r_8, b_{26}[8], b_{27}[8], b_{28}[8]) , (0,0) , -)

a_{21} = (X(r_9, b_{25}[8], b_{29}[8], b_{30}[8]) , (0,0) , -)

a_{22} = (X(r_{10}, b_{27}[8], b_{31}[8], b_{32}[8]) , (0,0) , -)

a_{23} = (J(b_{33}[8], b_{29}[8], b_{34}[8]) , 0, -)

a_{24} = (T(b_{34}[8], b_{33}[8]) , 0, -)

a_{25} = (J(b_{32}[8], b_{36}[8], b_{35}[8]) , 0, -)

a_{26} = (T(b_{35}[8], b_{36}[8]) , 0, -)

a_{27} = (Y(r_{11}, b_{30}[8], b_{31}[8], b_{37}[8]) , (0,0) , -)

a_{28} = (X(r_{12}, b_{37}[8], b_{38}[8], b_{39}) , (0,0) ,

[$M(r_{12})$ = 0 ($M(b_{38}[8])$:= $M(b_{37}[8])$):

T → ($M(b_{39})$:= 1)])

a_{29} = (X(r_{13}, b_{57}[8], b_{40}[8], b_{41}[8]) , (0,0) ,

[M(r_{13}) = 0 (M(b_{40}) := 1):

T → (M(b_{41}[8]) := M(b_{57}[8]))])

a_{30} = (Y(r_{14}, b_{39}, b_{40}, b_{42}) , (0,0) , -)

a_{31} = (F(b_{41}[8], b_{43}, b_{44}[8]) , 0,

[T → (M(b_{43}) := 1;

M(b_{44}[8]) := M(b_{41}[8]))])

a_{32} = (Y(r_{15}, b_{42}, b_{43}, b_{45}) , (0,0) , -)

a_{33} = (J(b_{48}, b_{45}, b_{46}) , 0, -)

a_{34} = (Y(r_{16}, b_{44}[8], b_{38}[8], Q_n[8]) , (0,0) , -)

a_{35} = (J(b_{46}, Q_n[8], b_{47}[8]) , 0,

[T → (M(b_{47}[8]) := M(b_{46}[8]))])

a_{36} = (F(b_{47}[8], b_{48}, b_{49}[8]) , 0,

[T → (M(b_{49}[8]) := M(b_{47}[8]);

M(b_{48}) := 1)])

a_{37} = (X(r_{17}, b_{49}[8], b_{50}[8], b_{51}[8]) , (1,2) ,

[M(r_{17}) = 0 (M(b_{50}(1)) := 1;

M(b_{50}(2)) := 0;

M(b_{50}(3)) := 1;

M(b_{50}(4)) := 2;

M(b_{50}(5)) := 0);

M(b_{50}(6)) := M(b_{49}(6));

$M(b_{50}(7)) := M(b_{49}(7));$

$M(b_{50}(8)) := M(b_{49}(8)) + 1):$

$T \rightarrow (M(b_{50}(6)) := M(b_{49}(6));$

$M(b_{50}(8)) := 2;$

$M(b_{50}(i)) := 0,$

$i = 1,2,3,4,5,7)])$

$a_{38} = (Y(r_{18}, b_{115}[8], b_{52}[8], b_{53}[8]) , (0,0) , -)$

$a_{39} = (Y(r_{19}, b_{53}[8], b_{51}[8], b_{54}[8]) , (0,0) , -)$

$a_{40} = (X(r_{20}, b_{54}[8], b_{55}[8], b_{56}[8]) , (0,0) , -)$

$a_{41} = (X (r_{21}, b_{56}[8], b_{57}[8], b_{58}[8]) , (0,0) , -)$

$a_{42} = (F(b_{113}[8], b_{59}[1], b_{60}[8]) , 0,$

$[T \rightarrow (M(b_{59}[1]) := 1;$

$M(b_{60}[8]) := M(b_{113}[8]))])$

$a_{43} = (J(b_{61}[1], b_{59}[1], b_{62}[1]), 0, -)$

$a_{44} = (F(b_{66}[8], b_{63}, b_{64}[8]) , 0,$

$[T \rightarrow (M(b_{63}) := 1;$

$M(b_{64}[8]) := M(b_{60}[8]))])$

$a_{45} = (T(b_{62}[1], b_{61}[1]) , 0, -)$

$a_{46} = (X(r_{22},b_{63}, b_{65}, b_{66}) , 0, -)$

$a_{47} = (J(b_{67}[1], b_{65}, b_{68}[1], 0,$

$[T \rightarrow (M(b_{68}(1)) := M(b_{67}(1)) + 1)])$

a_{48} = (T(b_{68}[1], b_{67}[1]) , 0, -)

a_{49} = (Y)r_{23}, b_{69}[8], b_{86}[8], b_{70}[8]) , (0,0) , -)

a_{50} = (Y(r_{24}, b_{70}[8], b_{72}[8], b_{71}[8]) , (0,0), -)

a_{51} = (Y(r_{25}, b_{64}[8], b_{71}[8], Q_p[8]) , (0,0) , -)

a_{52} = (X(r_{26}, Q_p[8], b_{73}[8], b_{72}[8]) , (0,0) , -)

a_{53} = (F(b_{73}[8],b_{74}[8], b_{75}[8]) , 0, -)

a_{54} = (J(b_{76}, b_{74}[8], b_{77}) , 0,

[T → (M(b_{17}) := 1;

END TASK(2) := CLOCK + M(b_{74}(4)))])

a_{55} = (T(b_{77}, b_{76}) , 0, -)

a_{56} = (T(b_{78} b_{79}) , 0, -)

a_{57} = (F(b_{79}, b_{78}, b_{80}) , 0, -)

a_{58} = (Y(r_{27}, b_{80}, b_{66}, b_{81}) , (0,0) , -)

a_{59} = (J(b_{81}, b_{75}[8], b_{82}[8]) , (0,0) ,

[(END TASK(2) > CLOCK) →

(M(b_{82}(4)) := END TASK(2) - CLOCK;

M(b_{82}(5)) := M(b_{75}(5)) + M(b_{77}(4))

-M(b_{82}(4));

M(b_{82}(i)) := M(b_{75}(i)) ,

i = 1,2,3,6,7,8):

T → (M(b_{82}(5)) := M(b_{75}(4)) + M(b_{75}(5));

M(b_{82}(4)) := 0;

M(b_{82}(i)) := M(b_{75}(i)) ,

i = 1,2,3,6,7,8)])

a_{60} = (X(r_{28}, b_{82}[8], b_{83}[8], b_{84}[8]) , (0,0) , -)

a_{61} = (X(r_{29}, b_{84}[8], b_{85}[8], b_{86}[8]) , (0,0) , -)

a_{62} = (X(r_{30}, b_{83}[8], b_{87}[8], b_{88}[8]) , (0,0) , -)

a_{63} = (X(r_{31}, b_{85}[8], b_{89}[8], b_{90}[8]) , (0,0) , -)

a_{64} = (J(b_{91}[8], b_{87}[8], b_{92}[8]) , 0, -)

a_{65} = (T(b_{42}[8], b_{91}[8]) , 0, -)

a_{66} = (J(b_{90}[8], b_{94}[8], b_{93}[8]) , 0, -)

a_{67} = (T(b_{93}[8], b_{94}]8]), 0, -)

a_{68} = (Y(r_{32}, b_{88}[8], b_{89}[8], b_{95}[8]) , (0,0) , -)

a_{69} = (X(r_{33}, b_{95}[8], b_{96}[8], b_{97}) , (0,0) ,

[M(r_{33}) = 0 → (M(b_{96}[8]) := M(b_{95}[8])):

T → (M(b_{97}) := 1)])

a_{70} = (Y(r_{34}, b_{107}[8], b_{111}[8], b_{98}[8]) , (0,0) ,

[M(b_{107}[8]) ≠ 0 → (M(b_{98}(7)) := 1;

M(b_{98}(i)) := M(b_{107}(i)) ,

i = 1,...,6,8):

T → (M(b_{98}(8)) := M(b_{111}(8));

M(b_{98}(i)) := 0 , 1 ≤ i ≤ 7)])

a_{71} = (F(b_{96}[8], r_{38}, b_{99}[8]) , 0,

[T → (M(r_{38}) := 1;

M(b_{99}(7)) := M(b_{96}(7));

M(b_{99}(i)) := 0 , i = 1,..., 6,8)])

a_{72} = (Y(R_{35}, $b_{99}[8]$, $b_{98}[8]$, $b_{100}[8]$) , (0,0) , -)

a_{73} = (J($b_{100}[8]$, $b_{101}[8]$, $b_{102}[8]$) , 0, -)

a_{74} = (X(r_{36}, $b_{102}[8]$, $b_{103}[8]$, $b_{104}[8]$) , (0,0) , -)

a_{75} = (F($b_{103}[8]$, $b_{115}[8]$, $b_{105}[8]$) , 0,

[T → (M(b_{105}(i)) := 0, 1 ≤ i ≤ 8;

M($b_{115}[8]$) := M($b_{103}[8]$))])

a_{76} = (Y(r_{37}, $b_{105}[8]$, $b_{104}[8]$, $b_{101}[8]$) , (0,0) , -)

a_{77} = (Y(r_{38}, b_{97}, b_{114}, b_{106}) , 0, -)

a_{78} = (J(b_{106}, b_{110}, r_{39}) , 0, -)

a_{79} = (X(r_{39}, $b_{57}[8]$, $b_{107}[8]$, $b_{108}[8]$) , (0,0) , -)

a_{80} = (F($b_{108}[8]$, $b_{109}[8]$, b_{110}) , 0,

[T → (M(b_{110}) := 1;

M($b_{109}[8]$) := M($b_{108}[8]$))])

a_{81} = (F($b_{109}[8]$, $b_{111}[8]$, $b_{112}[8]$) , 0,

[T → (M(b_{111}(8)) := M(b_{109}(8));

M(b_{111}(i)) := 0 , 1 ≤ i ≤ 7;

M(b_{112}(8)) := M(b_{109}(8)) + 2;

M(b_{112}(i)) := M(b_{109}(8)) , 1 ≤ i ≤ 7)])

$$a_{82} = (X(r_{40}, b_{112}[8], b_{113}[8], b_{114}) , (0,0) ,$$

$$[M(r_{40}) = 0 \rightarrow (M(b_{114}) := 1):$$

$$T \rightarrow (M(b_{113}(1)) := 1;$$

$$M(b_{113}(3)) := 1;$$

$$M(b_{113}(4)) := 1;$$

$$M(b_{113}(8)) := M(b_{112}(8));$$

$$M(b_{113}(i)) := 0 , i = 2,5,6,7)])$$

ξ = {END TASK[2] , CLOCK}

Ψ = {$\Psi(r_1)$,..., $\Psi(r_{40})$}

$$\Psi(r_1) = r_1 : [M(b_{17}(2)) = 1 \rightarrow M(r_1) := 1:$$

$$T \rightarrow M(r_1) := 0]$$

$$\Psi(r_2) = r_2 : [T \rightarrow M(r_2) := 1]$$

$$\Psi(r_3) = r_3 : [T \rightarrow M(r_3) := 1]$$

$$\Psi(r_4) = r_4 : [T \rightarrow M(r_4) := 0]$$

$$\Psi(r_5) = r_5 : [M(Q_{m-1}(3)) < M(Q_m(3))$$

$$M(r_5) := 1:$$

$$M(b_{17}[8]) \neq 0 \quad M(r_5) := 0]$$

$$\Psi(r_6) = r_6 : [M(b_8) = 1 \rightarrow M(r_6) := 1:$$

$$((M(b_{17}[8]) \neq 0) \quad ((\text{END TASK}(1) \leq \text{CLOCK})$$

$$(M(b_{17}(2)) - 1 \quad M(b_{17}(3))$$

$$\leq M(Q_m(3)))) \rightarrow M(r_6) := 0]$$

$$\Psi(r_7) = r_7 : [M(b_{24}(2)) = 1 \rightarrow M(r_7) := 1 :$$
$$T \rightarrow M(r_7) := 0]$$

$$\Psi(r_8) = r_8 : [M(b_{26}(4)) = 0 \rightarrow M(r_8) := 0 :$$
$$T \rightarrow M(r_8) := 1]$$

$$\Psi(r_9) = r_9 : [M(b_{25}(6)) > 0 \quad M(b_{25}(1)) = 1$$
$$\rightarrow M(r_9) := 1 :$$
$$T \rightarrow M(r_9) := 0]$$

$$\Psi(r_{10}) = r_{10} : [M(b_{27}(6)) > 0 \quad M(b_{27}(1)) = 1$$
$$\rightarrow M(r_{10}) := 0 :$$
$$T \rightarrow M(r_{10}) := 1]$$

$$\Psi(r_{11}) = r_{11} : [T \rightarrow M(r_{11}) := 0]$$

$$\Psi(r_{12}) = r_{12} : [M(b_{37}(1)) = 1 \rightarrow M(r_{12}) := 1 :$$
$$T \rightarrow M(r_{12}) := 0]$$

$$\Psi(r_{13}) = r_{13} : [M(b_{58}(7)) > 0 \rightarrow M(r_{13}) := 1 :$$
$$T \rightarrow M(r_{13}) := 0]$$

$$\Psi(r_{14}) = r_{14} : [T \rightarrow M(r_{14}) := 0]$$

$$\Psi(r_{15}) = r_{15} : [T \rightarrow M(r_{15}) := 0]$$

$$\Psi(r_{16}) = r_{16} : [T \rightarrow M(r_{16}) := 1]$$

$$\Psi(r_{17}) = r_{17} : [M(b_{49}(6)) > 0 \rightarrow M(r_{17}) := 1 :$$
$$T \rightarrow M(r_{17}) := 0]$$

$$\Psi(r_{18}) = r_{18} : [T \rightarrow M(r_{18}) := 0]$$

$\Psi(r_{19}) = r_{19}$:[T → $M(r_{19})$:= 0]

$\Psi(r_{20}) = r_{20}$:[T → $M(r_{20})$:= 0]

$\Psi(r_{21}) = r_{21}$:[$M(b_{56}(6)) > 0$ $M(b_{56}(7)) = 0$

→ $M(r_{21})$:= 0:

T → $M(r_{21})$:= 1]

$\Psi(r_{22}) = r_{22}$:[$M(b_{75}(2)) = 1$ $M(r_{22})$:= 1:

T → $M(r_{22})$:= 0]

$\Psi(r_{23}) = r_{23}$:[T → $M(r_{23})$:= 1]

$\Psi(r_{24}) = r_{24}$:[T → $M(r_{24})$:= 1]

$\Psi(r_{25}) = r_{25}$:[T → $M(r_{25})$:= 0]

$\Psi(r_{26}) = r_{26}$:[$M(Q_{p-1}(3)) < M(Q_p(3))$

→ $M(r_{26})$:= 1:

$M(b_{75}[8] \neq 0$ $M(r_{26})$:= 0]

$\Psi(r_{27}) = r_{27}$:[$M(b_{66}) = 1$ → $M(r_{27})$:= 1:

$((M(b_{17}[8]) \neq 0)$ ((END TASK(2) ≤ CLOCK)

$(M(b_{75}(2)) = 1$ $M(b_{17}(3))$

≤ $M(Q_p(3))))$ → $M(r_{27})$:= 0]

$\Psi(r_{28}) = r_{28}$:[$M(b_{82}(2)) = 1$ → $M(r_{28})$:= 1:

T → $M(r_{28})$:= 0]

$\Psi(r_{29}) = r_{29}$:[$M(b_{84}(4)) = 0$ → $M(r_{29})$:= 0:

T → $M(r_{29})$:= 1]

$$\Psi(r_{30}) = r_{30} : [M(b_{83}(6) > 0 \quad M(b_{83}(1)) = 1 \rightarrow M(r_{30}) := 1''$$

$$T \rightarrow M(r_{30}) := 0]$$

$$\Psi(r_{31}) = r_{31} : [M(b_{85}(6)) > 0 \quad M(b_{85}(1)) = 1 \rightarrow M(r_{31}) := 0:$$

$$T \rightarrow M(r_{31}) := 1]$$

$$\Psi(r_{32}) = r_{32} : [T \rightarrow M(r_{32}) := 0]$$

$$\Psi(r_{33}) = r_{33} : [M(b_{95}(1)) = 1 \rightarrow M(r_{33}) := 1:$$

$$T \rightarrow M(r_{33}) := 0]$$

$$\Psi(r_{34}) = r_{34} : [T \rightarrow M(r_{34}) := 0]$$

$$\Psi(r_{35}) = r_{35} : [T \rightarrow M(r_{35}) := 0]$$

$$\Psi(r_{37}) = r_{37} : [T \rightarrow M(r_{37}) := 0]$$

$$\Psi(r_{38}) = r_{38} : [T \rightarrow M(r_{38}) := 0]$$

$$\Psi(r_{40}) = r_{40} : [M(b_{112}(6)) > 1 \rightarrow M(r_{40}) := 0:$$

$$T \rightarrow M(r_{40}) := 1]$$

Appendix D: Token Flow in the Task Level E-Net Graph of the DAIS

An interpretation of the E-net graph in Figure 18 is begun by assigning an initial marking, M_0, to the net and initializing the environment variable CLOCK to zero. It is assumed that the net environment will: a) update CLOCK when no transitions are enabled and b) place tokens on locations b_{11} and b_{69}. Let M_0 be defined as follows:

$M(b_3) = M(b_9) = M(b_{61}) = M(b_{67}) = L[1]$, $L(1) = 0$;

$M(b_{33}) = M(b_{36}) = M(b_{91}) = M(b_{94}) = M(b_{101}) = J[8]$,

where $J(i) = 0$, $1 \leq i \leq 8$;

$M(b_{18}) = M(b_{21}) = M(b_{22}) = M(b_{46}) = M(b_{76}) = M(b_{79})$

$= M(b_{80}) = M(r_{39}) = 1$; $M(r_{36}) = 0$;

$M(b_j) = 0$, $1 \leq j \leq 115$, $j \neq 3, 9, 18, 21, 22, 33, 36, 46, 61, 67, 76, 79, 80, 91, 94, 101$;

$M(r_\ell) = \phi$, $1 \leq \ell \leq 40$, $\ell \neq 36, 39$.

Given M_0 and CLOCK = 0 , let the environment place token K[8] on location b_{11}, where

$K(1) = K(2) = K(5) = K(7) = K(8) = 0$;

$K(3) = 2$;

$K(4) = 4$;

$K(6) = 1$.

The following net operations will take place:

a_8 fires, $M(b_{12}) = K[8]$, $M(b_{11}) = 0$;

a_9 fires, $M(b_{13}) = K[8]$, $M(b_{12}) = 0$;

a_{10} fires, $M(Q_m) = K[8]$, $M(b_{13}) = 0$;

a_{11} fires, $M(b_{15}) = K[8]$, $M(Qm) = 0$;

a_{12} fires, $M(b_{16}) = M(b_{17}) = K[8]$, $M(b_{15}) = 0$;

a_{13} fires, $M(b_{19}) = K[8]$, $M(b_{16}) = M(b_{18}) = 0$;

a_{14} fires, $M(b_{18}) = K[8]$, $M(b_{19}) = 0$;

CLOCK = 4 ;

a_{17} fires, $M(b_{22}) = 0$, $M(b_{23}) = 1$;

a_{16} fires, $M(b_{22}) = M(b_{20}) = 1$, $M(b_{21}) = 0$;

a_{15} fires, $M(b_{21}) = 1$, $M(b_{20}) = 0$;

a_{18} fires, $M(b_{23}) = 0$, $M(b_{17}) = 0$,
$K(1) = K(2) = K(4) = K(7) = K(8) = 0$,
$K(3) = 2$, $K(5) = 4$, $K(6) = 1$,
$M(b_{23})$ = "updated" $K[8]$;

a_{19} fires, $M(b_{25}) = K[8]$, $M(b_{24}) = 0$;

a_{21} fires, $M(b_{30}) = K[8]$, $M(b_{25}) = 0$;

a_{27} fires, $M(b_{37}) = K[8]$, $M(b_{30}) - 0$;

a_{28} fires, $M(b_{38}) = K[8]$, $M(b_{37}) = 0$;

a_{34} fires, $M(Q_n) = K[8]$, $M(b_{38}) = 0$;

a_{35} fires, $M(b_{47}) = K[8]$, $M(Q_n) = 0$, $M(b_{46}) = 0$;

CLOCK = 6 ;

a_{36} fires, $M(b_{47}) = 0$, $M(b_{48}) = 1$,

$K(1) = K(2) = K(4) = K(7) = 0$,

$K(3) = K(8) = 2$, $K(5) = 4$, $K(6) = 1$,

$M(b_{49})$ = "updated" $K[8]$;

a_{37} fires, $M(b_{49}) = 0$, $K(6) = 1$, $K(8) = 2$,

$K(1) = K(2) = K(3) = K(4) = K(5) = K(7) = 0$,

$M(b_{51})$ = "updated" $K[8]$;

a_{39} fires, $M(b_{54}) = K[8]$, $M(b_{51}) = 0$;

a_{40} fires, $M(b_{56}) = K[8]$, $M(b_{54}) = 0$;

a_{41} fires, $M(b_{57}) = K[8]$, $M(b_{56}) = 0$;

a_{79} fires, $M(b_{108}) = K[8]$, $M(b_{57}) = 0$, $M(r_{41}) = 0$;

a_{80} fires, $M(b_{109}) = K[8]$, $M(b_{110}) = 1$, $M(b_{108}) = 0$;

a_{81} fires, $M(b_{112}) = K[8]$, $M(b_{109}) = 0$,

$K'(i) = 0$, $1 \leq i \leq 7$, $K'(8) = 2$;

$M(b_{111}) = K'[8]$;

a_{70} fires, $M(b_{98}) = K'[8]$, $M(b_{111}) = 0$;

a_{72} fires, $M(b_{100}) = K'[8]$, $M(b_{98}) = 0$;

a_{73} fires, $M(b_{102}) = K'[8]$, $M(b_{100}) = M(b_{101}) = 0$;

a_{74} fires, $M(b_{103}) = K'[8]$, $M(b_{102}) = 0$;

a_{75} fires, $M(b_{115}) = K'[8]$, $M(b_{103}) = 0$,

$M(b_{105}(i)) = 0$, $1 \leq i \leq 8$;

a_{76} fires, $M(b_{101}) = M(b_{105})$, $M(b_{105}) = 0$;

a_{38} fires, $M(b_{53}) = K'[8]$, $M(b_{115}) = 0$;

a_{39} fires, $M(b_{54}) = K'[8]$, $M(b_{53}) = 0$;

a_{40} fires, $M(b_{56}) = K'[8]$, $M(b_{54}) = 0$;

a_{41} fires, $M(b_{58}) = K'[8]$, $M(b_{56}) = 0$;

a_{29} fires, $M(b_{40}) = 1$, $M(b_{58}) = 0$;

a_{30} fires, $M(b_{42}) = 1$, $M(b_{42}) = 0$;

a_{32} fires, $M(b_{45}) = 1$, $M(b_{42}) = 0$;

a_{33} fires, $M(b_{46}) = 1$, $M(b_{45}) = M(b_{48}) = 0$;

CLOCK = 6 ;

a_{82} fires, $M(b_{114}) = 1$, $M(b_{112}) = 0$;

a_{77} fires, $M(b_{106}) = 1$, $M(b_{114}) = 0$;

a_{78} fires, $M(r_{39}) = 1$, $M(b_{106}) = 0$.

The net is now inactive until the environment places another token on b_{11} or b_{69}. The net marking now is identical to M_0.

Vita

Lieutenant Leslie A. Palmer was born April 23, 1949. After graduating from high school, Lt Palmer enlisted in the Navy, serving six years. He then attended the University of Texas at Austin and received a B.A. in Computer Science, magna cum laude. He was commissioned in the Air Force in May of 1978, and received a direct assignment to the Air Force Institute of Technology. He is a member of the IEEE Computer Society and president of the student chapter of the ACM at AFIT. Lt Palmer is married to the former Linda Alice Kneen of Belvidere, Vermont. They have one daughter, Heather Marie.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/70-12	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A GRAPH MODEL REPRESENTATION OF A DISTRIBUTED PROCESSOR COMPUTER SYSTEM		5. TYPE OF REPORT & PERIOD COVERED M.S. Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Leslie A. Palmer, 2 Lt, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December, 1979 13. NUMBER OF PAGES 147
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

JOSEPH P. HIPPS, Maj, USAF
Director of Public Affairs

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computer modeling
Graph models
Simulation
Computer systems analysis

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Three evaluation net graph models of the Digital Avionics Information System(DAIS) were constructed. The three models represent three increasingly lower levels of detail; the third model represents the DAIS at a task flow level of detail. The models are evaluated as analysis tools. Methods are presented for analyzing the DAIS structure and performance and examples are given. The biggest problem associated with a performance analysis using evaluation nets is the recording of data collected during

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*